eV+ Language Reference Guide, v2.0.x

This is a PDF/print version of the eV+ Language Reference Guide online documentation. A Table of Contents is provided so that you can locate the desired topics. Links to external documents will not work from this PDF file.

To view related eV+ manuals, you must access them from the eV+ menu. To return to the eV+ menu, click here.

> **NOTE:** Please see the eV+ Release Notes, which are included with your eV+ software, for a description of any recent changes.

Copyright Notice

The information contained herein is the property of Adept Technology, Inc., and shall not be reproduced in whole or in part without prior written approval of Adept Technology, Inc. The information herein is subject to change without notice and should not be construed as a commitment by Adept Technology, Inc. The documentation is periodically reviewed and revised.

Adept Technology, Inc., assumes no responsibility for any errors or omissions in the documentation. Critical evaluation of the documentation by the user is welcomed. Your comments assist us in preparation of future documentation. Please submit your comments to: techpubs@adept.com.

Copyright © 1994 - 2013 by Adept Technology, Inc. All rights reserved.

Adept, the Adept logo, the Adept Technology logo, AdeptVision, AIM, Blox, Bloxview, FireBlox, Fireview, Meta Controls, MetaControls, Metawire, Soft Machines, and Visual Machines are registered trademarks of Adept Technology, Inc.

Brain on Board is a registered trademark of Adept Technology, Inc. in Germany.

Adept ACE, ACE PackXpert, Adept SmartController EX, and eV+ are trademarks of Adept Technology, Inc.

Any trademarks from other companies used in this publication are the property of those respective companies.

Created in the United States of America

Table Of Contents

Introduction **15**
Compatibility 16
Licenses 16
Related Publications 16
Dangers, Warnings, Cautions, and Notes in Manual 17
Conventions 18
How Can I Get Help? 21

Keyword Overview **23**
New or Enhanced Keywords 24
eV+ Language Quick Reference 25

Keyword Descriptions **41**
Descriptions of eV+ Keywords 42
Documentation Conventions for Keywords 42
ABORT program instruction 45
ABOVE program instruction 47
ABS real-valued function 48
ACCEL program instruction 49
ACCEL real-valued function 52
ALIGN program instruction 54
ALWAYS keyword 55
AND operator 56
ANY program instruction 57
APPRO program instruction 58
APPRO program instruction 60
ASC real-valued function 62
ATAN2 real-valued function 64
ATTACH program instruction 65
AUTO program instruction 71
AUTO.POWER.OFF system switch 74
BAND operator 76
BASE program instruction 78
BASE transformation function 80

BCD real-valued function 81
BELOW program instruction 82
BELT real-valued function 83
BELT system switch 85
BELT.MODE system parameter 87
BITS program instruction 89
BITS real-valued function 91
BMASK real-valued function 93
BOR operator 94
BRAKE program instruction 96
BREAK program instruction 97
BSTATUS real-valued function 98
BXOR operator 100
BY keyword 102
CALIBRATE program instruction 103
CALL program instruction 106
CALLP program instruction 109
CALLS program instruction 111
CAS real-valued function 113
CASE program instruction 115
$CHR string function 118
CLEAR.EVENT program instruction 119
CLOSE and CLOSEI program instruction 120
CLOSE and CLOSEI program instruction 122
COARSE program instruction 124
COM operator 126
CONFIG real-valued function 127
COS real-valued function 133
CP system switch 134
CPOFF program instruction 135
CPON program instruction 137
CYCLE.END program instruction 139
DBLB real-valued function 141
$DBLB string function 143
DCB real-valued function 144
DECEL.100 system switch 145

$DECODE string function 146
DECOMPOSE program instruction 149
$DEFAULT string function 151
DEFBELT program instruction 153
DEFINED real-valued function 156
DELAY program instruction 158
DELAY.IN.TOL system switch 160
DELAY.POWER.OFF system switch 161
DEPART and DEPARTS program instruction 162
DEPART and DEPARTS program instruction 164
DEST transformation function 166
DETACH program instruction 168
DEVICE program instruction 170
DEVICE real-valued function 172
DEVICES program instruction 174
DISABLE program instruction 176
DISTANCE real-valued function 178
DN.RESTART program instruction 179
DO program instruction 180
DOS program instruction 182
DRIVE program instruction 184
DRY.RUN system switch 186
DURATION program instruction 188
DURATION real-valued function 191
DX, DY, DZ real-valued function 193
DX, DY, DZ real-valued function 195
DX, DY, DZ real-valued function 197
ELSE program instruction 199
ENABLE program instruction 200
$ENCODE string function 202
END program instruction 205
.END keyword 206
ERROR real-valued function 207
$ERROR string function 211
ESTOP program instruction 212
EXECUTE program instruction 213

EXIT program instruction 217
FALSE real-valued function 219
FCLOSE program instruction 220
FCMND program instruction 222
FCOPY program instruction 232
FDELETE program instruction 234
FEMPTY program instruction 236
FINE program instruction 238
FLIP program instruction 240
FLTB real-valued function 243
$FLTB string function 245
FOPEN program instruction 246
FOPEN_ program instruction 249
FOR program instruction 253
FRACT real-valued function 255
FRAME transformation function 256
FREE real-valued function 258
FSEEK program instruction 260
FSET program instruction 262
GETC real-valued function 265
GET.EVENT real-valued function 267
GLOBAL program instruction 268
GOTO program instruction 270
HALT program instruction 272
HAND real-valued function 273
HAND.TIME system parameter 274
HERE program instruction 276
HERE transformation function 278
ID real-valued function 279
$ID string function 285
IDENTICAL real-valued function 286
IF logical_expr THEN program instruction 287
IF logical_expr GOTO program instruction 289
IGNORE program instruction 291
INRANGE real-valued function 292
INSTALL program instruction 295

INT real-valued function 296
INTB real-valued function 297
$INTB string function 299
INVERSE transformation function 300
IOSTAT real-valued function 302
IPS keyword 305
JHERE program instruction 306
JMOVE program instruction 307
JOG program instruction 308
KILL program instruction 312
LAST real-valued function 313
LATCH transformation function 315
LATCHED real-valued function 316
LEFTY program instruction 318
LEN real-valued function 320
LNGB real-valued function 321
$LNGB string function 323
LOCAL program instruction 325
LOCK program instruction 327
MAX real-valued function 329
MC program instruction 330
MCS program instruction 332
MESSAGES system switch 334
$MID string function 335
MIN real-valued function 336
MMPS keyword 337
MOD operator 338
MOVE and MOVES program instruction 339
MOVEC program instruction 341
MOVE and MOVES program instruction 349
MULTIPLE program instruction 351
NETWORK real-valued function 353
NEXT program instruction 355
NOFLIP program instruction 357
NONULL program instruction 358
NOOVERLAP program instruction 360

NORMAL transformation function 362
NOT operator 363
NOT.CALIBRATED system parameter 364
NULL program instruction 366
NULL transformation function 368
OFF real-valued function 369
ON real-valued function 370
OPEN program instruction 371
OPEN program instruction 373
OR operator 375
OUTSIDE real-valued function 377
OVERLAP program instruction 378
PACK program instruction 380
PANIC program instruction 382
PARAMETER program instruction 383
PARAMETER real-valued function 385
PAUSE program instruction 387
#PDEST precision-point function 388
PDNT.CLEAR program instruction 388
PDNT.NOTIFY program instruction 389
PDNT.WRITE program instruction 390
#PHERE precision-point function 394
PI real-valued function 395
#PLATCH precision-point function 396
POS real-valued function 397
POWER system switch 398
#PPOINT precision- point function 400
PRIORITY real-valued function 402
.PROGRAM program instruction 403
PROMPT program instruction 406
RANDOM real-valued function 408
REACT program instruction 409
REACTE program instruction 412
REACTI program instruction 414
READ program instruction 416
READY program instruction 420

RELAX and RELAXI program instruction 422
RELAX and RELAXI program instruction 424
RELEASE program instruction 426
RESET program instruction 427
RETURN program instruction 428
RETURNE program instruction 429
RIGHTY program instruction 430
ROBOT system switch 431
ROBOT.OPR program instruction 433
ROBOT.OPR real-valued function 439
RUNSIG program instruction 441
RX, RY, RZ transformation functions 443
RX, RY, RZ transformation functions 444
RX, RY, RZ transformation functions 445
SCALE transformation function 446
SCALE.ACCEL system switch 447
SCALE.ACCEL.ROT system switch 449
SELECT program instruction 450
SELECT real-valued function 453
SET program instruction 455
SET.EVENT program instruction 457
#SET.POINT precision point function 458
SETBELT program instruction 459
SETDEVICE program instruction 461
SHIFT transformation function 463
SIG real-valued function 464
SIG.INS real-valued function 466
SIGN real-valued function 468
SIGNAL program instruction 469
SIN real-valued function 471
SINGLE program instruction 472
SOLVE.ANGLES program instruction 474
SOLVE.FLAGS real-valued function 481
SOLVE.TRANS program instruction 483
SPEED program instruction 485
SPEED real-valued function 488

SQR real-valued function 490
SQRT real-valued function 491
STATE real-valued function 492
STATUS real-valued function 501
STOP program instruction 503
STRDIF real-valued function 505
SWITCH program instruction 507
SWITCH real-valued function 509
$SYMBOL string function 511
SYMBOL.PTR real-valued function 512
$SYS.INFO string function 513
TAS real-valued function 514
TASK real-valued function 517
TIME program instruction 520
TIME real-valued function 522
$TIME string function 525
$TIME4 string function 527
TIMER program instruction 529
TIMER real-valued function 530
TOOL program instruction 533
TOOL transformation function 534
TPS real-valued function 535
TRANS transformation function 536
$TRANSB string function 538
TRANSB transformation function 539
TRUE real-valued function 541
$TRUNCATE string function 542
TYPE program instruction 543
$UNPACK string function 546
UNTIL program instruction 548
UPPER system switch 549
VAL real-valued function 551
VALUE program instruction 552
WAIT program instruction 553
WAIT.EVENT program instruction 555
WHILE program instruction 558

WINDOW program instruction 560
WINDOW real-valued function 562
WRITE program instruction 566
XOR operator 569

ID Option Words 571
Introduction to ID Option Words 572
Robot Option Words 572
System Option Words 574
Processor Option Word 576

System Messages 577
Introduction to System Messages 578
System Messages - Alphabetical List 578
SystemMessages - Numerical List 653

Introduction

The following topics are described in this chapter:

Compatibility **16**
Licenses **16**
Related Publications **16**
Dangers, Warnings, Cautions, and Notes in Manual **17**
Conventions **18**
How Can I Get Help? **21**

Compatibility

This guide is for use with eV+ systems version 2.0.x and later. This guide provides reference material and descriptions of keywords for the eV+ programming language. For information on the eV+ operating system and descriptions of the monitor commands, see the *eV+ Operating System User's Guide* and the *eV+ Operating System Reference Guide.*

See the *eV+ Release Notes* for a summary of changes for each version.

Licenses

The eV+ Operating System software requires the appropriate "eV+ version license" to be installed in the nonvolatile RAM (NVRAM) on the Adept controller's system processor board. If this license is not installed, the system displays a *Protection Error* after attempting to boot the controller.

The Adept controller is supplied with the Adept network licenses installed. This allows you to connect your Adept controller to an Ethernet network and to share files on it. You can use Adept ACE to share files between your PC and controller.

Related Publications

This reference guide is a companion to the *eV+ Language User's Guide*, which covers the principles of the eV+ programming language and robot-control system.

In addition to being a complete programming language, eV+ is also a complete operating system that controls equipment connected to Adept controllers. The *eV+ Operating System User's Guide* and *eV+ Operating System Reference Guide* detail the eV+ operating system. You must be familiar with the operating system in order to effectively use the eV+ programming language.

The most current releases of some related publications may be for an earlier version of the eV+ system. You need to use them in conjunction with the release notes published since those books were published.

You may also need to refer to one or more of the manuals listed in the following table.

Manual	Material Covered
eV+Release Notes	Late-breaking changes not in manuals; summary of changes.
Adept ACE User's Guide	Describes the Adept ACE user interface, which is used for configuration, control, and programming of the Adept robot system.
AdeptSight User's Guide	Describes the interface, use, and programming of the optional AdeptSight vision system.

Manual	Material Covered
AdeptSight Reference Guide	Describes the AdeptSight vision commands used for custom vision programming.
Adept SmartController User's Guide	Instructions for setting up, configuring, and maintaining the controller on which eV+ runs.
Robot or motion device user's guides (if connected to your system)	Instructions for installing and maintaining the motion device connected to your system.

Dangers, Warnings, Cautions, and Notes in Manual

There are four levels of special notation used in this manual. In descending order of importance, they are:

DANGER: This indicates an imminently hazardous electrical situation which, if not avoided, will result in death or serious injury.

DANGER: This indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

WARNING: This indicates a potentially hazardous electrical situation which, if not avoided, could result in serious injury or major damage to the equipment.

WARNING: This indicates a potentially hazardous situation which, if not avoided, could result in serious injury or major damage to the equipment.

CAUTION: This indicates a situation which, if not avoided, could result in minor injury or damage to the equipment.

NOTE: This provides supplementary information, emphasizes a point or procedure, or gives a tip for easier operation.

Conventions

Typographic Conventions

The following typographic conventions are used throughout this manual:

This	Represents
ALL CAPITALS	eV+ file names, directory names, commands, keywords, and attributes; also acronyms. A *physical* key or button that you must press, such as the Y, N, and ENTER keys.
`monospace`	Monitor displays and code examples.
bold **bold**/regular	Bold type is used for subroutine names, variable names, and program names, such as **a.diskcopy**. Bold type also is used for window items that you choose and window items that do not have initial capital letters in all principal words. In a typing or entering instructions, anything that you must type exactly as it appears. For example, if you are asked to type **execute 1 a.diskcopy**, you type all the bold characters exactly as they are printed. What you type is shown in lowercase letters unless it must be typed in uppercase letters to work properly. You may always substitute a currently valid shortcut form when typing a eV+ command. In order for the eV+ system to process your typing, you must conclude your entry by pressing the ENTER or RETURN key.

	In Syntax, place holders, in formal syntax definitions, for information that you provide. You must replace such a place holder written in ***bold*** weight but need not replace an optional one, which is written in *regular* weight.
italics	Indicates new terms and other emphasized words.
Initial Capitals	The name of an object such as a window, screen, menu, dialog box, or dialog box component. Examples are the Display menu and the Task Profiler window.
"Quotation marks"	Menu items, prompts, or any literal text that is being referenced.

Keyboard Conventions

Key combinations appear in the following format:

Notation	Meaning
KEY1+KEY2	A plus sign (+) between keys means that you must press and hold down KEY1, then press KEY2. For example, "Press CTRL+Z" means that you press CTRL and hold it down while you press Z.

Selecting, Choosing, and Pressing Items

In a context using windows, the terms *select, choose,* and *press* have different and specific meanings. Selecting an item usually means marking or highlighting it, as in picking a radio button. Selecting alone does not initiate an action.

Choosing an item carries out an action. For example, choosing a menu item may open a window or carry out a command. You can also initiate an action by choosing a command button (a push button or a standard button). You often must select an item before you can choose it.

Often you can use a combination of keyboard and mouse techniques for selecting and choosing.

Pressing refers to *physical* buttons or keys. For example, you press the save key or press the ENTER key. By contrast, you select or choose a window button.

Values, Variables, and Expressions

The parameters to eV+ keywords can generally be satisfied with a specific value of the correct data type, a variable of the correct data type, or an expression that resolves to the correct type. Unless specifically stated, parameters can be replaced with a value, variable, or expression (of the correct type). The most common case where a parameter cannot be satisfied with all three options occurs when data is being returned in one of the parameters. In this case, a variable must be used; the parameter description states this restriction.

Integers and Real Values

In eV+ integers and real values are not different data types. Real values satisfy parameters requiring integers by rounding the real value. Where real values are required, an integer is considered a special case of a real value with no fractional part.

Special Notation

Numbers shown in other than decimal format are preceded with a carat (^) and the letter H for hexadecimal or B for binary, and with just a carat for Octal. For example, ^HF = ^B1111 = ^17 = 15.

How Can I Get Help?

For details on getting assistance with your Adept software or hardware, you can access the following information sources on the Adept corporate website:

- For Contact information: http://www.adept.com/contact/americas
- For Product Support information: http://www.adept.com/support/service-and-support/main
- For user discussions, support, and programming examples: http://www.adept.com/forum/
- For further information about Adept Technology, Inc.: http://www.adept.com

Keyword Overview

The following topics are described in this chapter:

New or Enhanced Keywords **24**
eV+ Language Quick Reference **25**

New or Enhanced Keywords

For information on new or enhanced keywords listed by eV+ software release, select a link below:

eV+ 2.0.x Release Notes

eV+ Language Quick Reference

This Quick Reference table is arranged alphabetically by command name, click an underlined letter to jump to the first command that begins with that letter.

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Type	Description
K	Keyword
O	Operator
PI	Program instruction
RF	Real-valued function
SW	System switch
SP	System parameter
TF	Transformation function
ST	String function
PP	Precision point

Keyword	Type	Description
ABORT	PI	Terminate execution of an executing program task.
ABOVE	PI	Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.
ABS	RF	Return absolute value.
ACCEL	PI	Set acceleration and deceleration for robot motions. Optionally, specify a defined acceleration profile.
ACCEL	RF	Return the current setting for robot acceleration or deceleration setting or return the maximum allowable percentage limits defined in the robot configuration profile.
ALIGN	PI	Align the robot tool Z axis with the nearest world axis.
ALWAYS	K	Used with certain program instructions to specify a long-term effect.
AND	O	Perform the logical AND operation on two values.
ANY	PI	Signal the beginning of an alternative group of instructions for the

Keyword	Type	Description
		CASE structure.
APPRO APPROS	PI	Start a robot motion toward a location defined relative to specified location.
ASC	RF	Return an ASCII character value from within a string.
ATAN2	RF	Return the size of the angle (in degrees) that has its trigonometric tangent equal to value_1/value_2.
ATTACH	PI	Make a device available for use by the application program.
AUTO	PI	Declare temporary variables that are automatically created on the program stack when the program is entered.
AUTO.POWER.OFF	SW	Control whether or not eV+ disables high power when certain motion errors occur.
BAND	O	Perform the *binary* AND operation on two values.
BASE	PI	
BASE	TF	Return the transformation value that represents the translation and rotation set by the last BASE command or instruction.
BCD	RF	Convert a real value to Binary Coded Decimal (BCD) format.
BELOW	PI	Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.
BELT	SW	Control the function of the conveyor tracking features of the eV+ system.
BELT	RF	Return information about a conveyor belt being tracked with the conveyor tracking feature.
BELT.MODE	SP	Set characteristics of the conveyor tracking feature of the eV+ system.
BITS	PI	Set or clear a group of digital signals based on a value.
BITS	RF	Read multiple digital signals and return the value corresponding to the binary bit pattern present on the signals.
BMASK	RF	Create a bit mask by setting individual bits.
BOR	O	Perform the binary OR operation on two values.
BRAKE	PI	Abort the current robot motion.
BREAK	PI	Suspend program execution until the current motion completes.
BSTATUS	RF	Return information about the status of the conveyor tracking

Keyword	Type	Description
		system.
BXOR	O	Perform the binary exclusive-OR operation on two values.
BY	K	Complete the syntax of the SCALE and SHIFT functions.
CALIBRATE	PI	Initialize the robot positioning system with the robot's current position.
CALL	PI	Suspend execution of the current program and continue execution with a new program (that is, a subroutine).
CALLP	PI	Call a program given a pointer to the program in memory.
CALLS	PI	Suspend execution of the current program and continue execution with a new program (that is, a subroutine) specified with a string value.
CAS	RF	Compare a real variable to a test value, and conditionally sets a new value as one indivisible operation.
CASE	PI	Initiate processing of a CASE structure by defining the value of interest.
$CHR	ST	Return a one-character string corresponding to a given ASCII value.
CLEAR.EVENT	PI	Clear an event associated with the specified task.
CLOSE CLOSEI	PI	Close the robot gripper.
COARSE	PI	Enable a low-precision feature of the robot hardware servo.
COM	O	Perform the binary complement operation on a value.
CONFIG	RF	Return a value that provides information about the robot's geometric configuration, or the status of the motion servo-control features.
COS	RF	Return the trigonometric cosine of a given angle.
CP	SW	Control the continuous-path feature.
CPOFF	PI	Instruct the eV+ system to stop the robot at the completion of the next motion instruction (or all subsequent motion instructions) and null position errors.
CPON	PI	Instruct the eV+ system to execute the next motion instruction (or all subsequent motion instructions) as part of a continuous path.
CYCLE.END	PI	Terminate the executing program in the specified task the next time it executes a STOP program instruction (or its equivalent).

Keyword	Type	Description
		Suspend processing of an executable program until a program running in the specified task completes execution.
DBLB	RF	Return the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.
$DBLB	ST	Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.
DCB	RF	Convert BCD digits into an equivalent integer value.
$DECODE	ST	Extract part of a string as delimited by given break characters.
DECOMPOSE	PI	Extract the (real) values of individual components of a location value.
$DEFAULT	ST	Return a string containing the current system default device, unit, and directory path for disk file access.
DEFBELT	PI	Define a belt variable for use with a conveyor tracking robot.
DEFINED	RF	Determine whether a variable has been defined.
DELAY	PI	Cause robot motion to stop for the specified time.
DELAY.IN.TOL	SW	Controls the timing of COARSE or FINE nulling after eV+ completes a motion segment.
DELAY.POWER.OFF	SW	Enable/disable the ESTOP timer delay feature for servo errors.
DEPART DEPARTS	PI	Start a robot motion away from the current location.
DEST	TF	Return a transformation value representing the planned destination location for the current robot motion.
DETACH	PI	Release a specified device from the control of the application program.
DEVICE	PI	Send a command or data to an external device and, optionally, return data back to the program. (The actual operation performed depends on the device referenced.)
DEVICE	RF	Return a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.
DEVICES	PI	Send commands or data to an external device and optionally return data. The actual operation performed depends on the device referenced.
DISABLE	PI	Turn off one or more system control switches.
DISTANCE	RF	Determine the distance between the points defined by two location

Keyword	Type	Description
		values.
DN.RESTART	PI	Restarts DeviceNet communication if the CanBus goes offline.
DO	PI	Introduce a DO program structure.
DOS	PI	Execute a program instruction defined by a string expression.
DRIVE	PI	Move an individual joint of the robot.
DRY.RUN	SW	Control whether or not eV+ communicates with the robot.
DURATION	PI	Set the minimum execution time for subsequent robot motions.
DURATION	RF	Return the current setting of one of the motion DURATION specifications.
DX DY DZ	RF	Return a displacement component of a given transformation value.
ELSE	PI	Separate the alternate group of statements in an IF ... THEN control structure.
ENABLE	PI	Turn on one or more system control switches.
$ENCODE	ST	Return a string created from output specifications. The string produced is similar to the output of a TYPE instruction.
END	PI	Mark the end of a control structure.
.END	PI	Mark the end of a eV+ program.
ERROR	RF	Return the error number of a recent error that caused program execution to stop or caused a REACTE reaction.
ESTOP	PI	Assert the emergency-stop signal to stop the robot.
EXECUTE	PI	Begin execution of a control program.
EXIT	PI	Branch to the statement following the nth nested loop of a control structure.
FALSE	RF	Return the value used by eV+ to represent a logical false result.
FCLOSE	PI	Close the disk file, graphics window, or graphics icon currently open on the specified logical unit.
FCMND	PI	Generate a device-specific command to the input/output device specified by the logical unit.

Keyword	Type	Description
FCOPY	PI	Copy the information in an existing disk file to a new disk file.
FDELETE	PI	Delete the specified disk file, the specified graphics window and all its child windows, or the specified graphics icon.
FEMPTY	PI	Empty any internal buffers in use for a disk file or a graphics window by writing the buffers to the file or window if necessary.
FINE	PI	Enable a high-precision feature of the robot hardware servo.
FLIP	PI	Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.
FLTB	RF	Return the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.
$FLTB	ST	Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.
FOPEN	PI	Create and open a new graphics window or TCP connection, or open an existing graphics window for subsequent input or output.
FOPENA FOPEND FOPENR FOPENW	PI	Open a disk file for read-only, read-write, read-write-append, or read-directory, as indicated by the last letter of the instruction name.
FOR	PI	Execute a group of program instructions a certain number of times.
FRACT	RF	Return the fractional part of the argument.
FRAME	TF	Return a transformation value defined by four positions.
FREE	RF	Return the amount of unused free memory storage space.
FSEEK	PI	Position a file open for random access and initiate a read operation on the specified record.
FSET	PI	Set or modify attributes of a graphics window, serial line, or network device.
GETC	RF	Return the next character (byte) from a device or input record on the specified logical unit.
GET.EVENT	RF	Return events that are set for the specified task.
GLOBAL	PI	Declare a variable to be global and specify the type of the variable.
GOTO	PI	Perform an unconditional branch to the program step identified by the given label.
HALT	PI	Stop program execution and do not allow the program to be

Keyword	Type	Description
		resumed.
HAND	RF	Return the current hand opening.
HERE	PI	Set the value of a transformation or precision-point variable equal to the current robot location.
HERE	TF	Return a transformation value that represents the current location of the robot tool point.
ID	RF	Return values that identify the configuration of the current system.
$ID	ST	Return the system ID string.
IDENTICAL	RF	Determine whether two location values are exactly the same.
IF... GOTO	PI	Branch to the specified step label if the value of the logical expression is TRUE (nonzero).
IF ... THEN	PI	Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.
IGNORE	PI	Cancel the effect of a REACT or REACTI instruction.
INRANGE	RF	Return a value that indicates whether a location can be reached by the robot and, if not, why not.
INSTALL	PI	Install or remove software options available to Adept systems.
INT	RF	Return the integer part of the value.
INTB	RF	Return the value of two bytes of a string interpreted as a signed 16-bit binary integer.
$INTB	ST	Return a 2-byte string containing the binary representation of a 16-bit integer.
INVERSE	TF	Return the transformation value that is the mathematical inverse of the given transformation value.
IOSTAT	RF	Return status information for the last input/output operation for a device associated with a logical unit.
IPS	K	Specify the units for a SPEED instruction as inches per second.
JHERE	PI	Records the current robot joint positions in real or double-precision variables. This instruction supports MicroeV+.
JMOVE	PI	Moves all robot joints to positions described by a list of joint values. The robot performs a coordinated motion in joint-interpolated mode. This instruction supports MicroeV+.
JOG	PI	Moves ("jogs") the specified axis or joint of the robot. Each time JOG

Keyword	Type	Description
		executes, the specified axis or joint moves for 200 ms.
KILL	PI	Clear a program execution stack and detach any I/O devices that are attached.
LAST	RF	Return the highest index used for an array (dimension).
LATCH	TF	Return a transformation value representing the location of the robot at the occurrence of the last external trigger.
LATCHED	RF	Return the status of the external trigger.
LEFTY	PI	Request a change in the robot configuration during the next motion so that the first two links of a SCARA robot resemble a human's left arm.
LEN	RF	Return the number of characters in the given string.
LNGB	RF	Return the value of four bytes of a string interpreted as a signed 32-bit binary integer.
$LNGBLOCK	ST	Set the program reaction lock-out priority to the value given.
MAX	RF	Return the maximum value contained in the list of values.
MC	PI	Introduce a monitor command within a command program.
MCS	PI	Invoke a monitor command from an application program.
MESSAGES	SW	Enable or disable output to the system terminal from TYPE instructions.
$MID	ST	Return a substring of the specified string.
MIN	RF	Return the minimum value contained in the list of values.
MMPS	K	Specify the units for a SPEED instruction as millimeters per second.
MOD	O	Compute the modulus of two values.
MOVE MOVES	PI	Initiate a robot motion to the position and orientation described by the given location.
MOVEC	PI	Initiate a circular/arc-path robot motion using the positions and orientations described by the given locations.
MULTIPLE	PI	Allow full rotations of the robot wrist joints.
NETWORK	RF	Return network status and IP address information
NEXT	PI	Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.

Keyword	Type	Description
NOFLIP	PI	Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a positive value.
NONULL	PI	Instruct the eV+ system not to wait for position errors to be nulled at the end of continuous-path motions.
NOOVERLAP	PI	Generate a program error if a motion is planned that causes selected multiturn axes to turn more than around) in order to avoid a limit stop.
NORMAL	TF	Correct a transformation for any mathematical round-off errors.
NOT	PI	Perform logical negation of a value.
NOT.CALIBRATED	SP	Indicate (or assert) the calibration status of the robots connected to the system.
NULL	PI	Instruct the eV+ system to wait for position errors to be nulled at the end of continuous path motions.
NULL	TF	Return a null transformation value-one with all zero components.
OFF	RF	Return the value used by eV+ to represent a logical false result.
ON	RF	Return the value used by eV+ to represent a logical true result.
OPEN OPENI	PI	Open the robot gripper.
OR	PI	Perform the *logical* OR operation on two values.
OUTSIDE	RF	Test a value to see if it is outside a specified range.
OVERLAP	PI	Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.
PACK	PI	Replace a substring within an array of (128-character) string variables, or within a (nonarray) string variable.
PANIC	PI	Simulate an external E-stop or panic button press; stop all robots immediately, but do not turn off HIGH POWER.
PARAMETER	PI	Set the value of a system parameter.
PARAMETER	RF	Return the current setting of the named system parameter.
PAUSE	PI	Stop program execution but allow the program to be resumed.
#PDEST	PP	Return a precision-point value representing the planned destination location for the current robot motion.
PDNT.CLEAR	PI	Clears the current notification or custom message window, if any,

Keyword	Type	Description
		and returns the T20 pendant back to the Home screen.
PDNT.NOTIFY	PI	Creates a pendant notification.
PDNT.WRITE	PI	Sets the pendant's Custom Message screen.
#PHERE	PP	Return a precision-point value representing the current location of the currently selected robot.
PI	RF	Return the value of the mathematical constant pi (3.141593).
#PLATCH	PP	Return a precision-point value representing the location of the robot at the occurrence of the last external trigger.
POS	RF	Return the starting character position of a substring in a string.
POWER	SP	Control or monitor the status of high power.
#PPOINT	PP	Return a precision-point value composed from the given components.
PRIORITY	RF	Return the current reaction lock-out priority for the program.
.PROGRAM	PI	Define the arguments that are passed to a program when it is invoked.
PROMPT	PI	Display a string on the system terminal and wait for operator input.
RANDOM	RF	Return a pseudorandom number.
REACT	PI	Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.
REACTE	PI	Initiate the monitoring of errors that occur during execution of the current program task.
REACTI	PI	Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal transitions properly and optionally trigger a subroutine call.
READ	PI	Read a record from an open file or from an attached device that is not file oriented. For a network device, read a string from an attached and open TCP connection.
READY	PI	Move the robot to the READY location above the workspace, which forces the robot into a standard configuration.
RELAX RELAXI	PI	Limp the pneumatic hand.

Keyword	Type	Description
RELEASE	PI	Allow the next available program task to run.
RESET	PI	Turn off all the external output signals.
RETURN	PI	Terminate execution of the current subroutine, and resume execution of the suspended program at its next step. A program may have been suspended by issuing a CALL, CALLP, or CALLS instruction, or by the triggering of a REACT, REACTE, or REACTI condition.
RETURNE	PI	Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the instruction that caused the subroutine to be invoked.
RIGHTY	PI	Request a change in the robot configuration during the next motion so that the first two links of the robot resemble a human's right arm.
ROBOT	SW	Enable or disable one robot or all robots.
ROBOT.OPR	PI	Execute operations that are specific to the currently selected robot or robot module.
ROBOT.OPR	RF	Returns robot-specific data for the currently selected robot.
RUNSIG	PI	Turn on (or off) the specified digital signal as long as execution of the invoking program task continues.
RX RY RZ	TF	Return a transformation describing a rotation.
SCALE	TF	Return a transformation value equal to the transformation parameter with the position scaled by the scale factor.
SCALE.ACCEL	SW	Enable or disable the scaling of acceleration and deceleration as a function of program speed, as long as the program speed is below a preset threshold.
SCALE.ACCEL.ROT	SW	Specify whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.
SELECT	PI	Select a unit of the named device for access by the current task.
SELECT	RF	Return the unit number that is currently selected by the current task for the device named.
SET	PI	Set the value of the location variable on the left equal to the location value on the right of the equal sign.
SET.EVENT	PI	Set an event associated with the specified task.

Keyword	Type	Description
#SET.POINT	PP	Return the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.
SETBELT	PI	Set the encoder offset of the specified belt variable equal to the value of the expression.
SETDEVICE	PI	Initialize a device or set device parameters. (The actual operation performed depends on the device referenced.)
SHIFT	TF	Return a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.
SIG	RF	Returns the logical AND of the states of the indicated digital signals.
SIG.INS	RF	Return an indication of whether a digital I/O signal is installed in the system, or whether a software signal is available in the system.
SIGN	RF	Return the value 1, with the sign of the value parameter.
SIGNAL	PI	Turn on or off external digital output signals or internal software signals.
SIN	RF	Return the trigonometric sine of a given angle.
SINGLE	PI	Limit rotations of the robot wrist joint to the range -180 degrees to +180 degrees.
SOLVE.ANGLES	PI	Compute the robot joint positions (for the current robot) that are equivalent to a specified transformation.
SOLVE.FLAGS	RF	Return bit flags representing the robot configuration specified by an array of joint positions.
SOLVE.TRANS	PI	Compute the transformation equivalent to a given set of joint positions for the current robot.
SPEED	PI	Set the nominal speed for subsequent robot motions.
SPEED	RF	Return one of the system motion speed factors.
SQR	RF	Return the square of the parameter.
SQRT	RF	Return the square root of the parameter.
STATE	RF	Return a value that provides information about the robot system state.
STATUS	RF	Return status information for an application program.
STOP	PI	Terminate execution of the current program cycle.
STRDIF	RF	Compare two strings byte by byte for the purpose of sorting. This function always compares bytes exactly. It ignores the setting of the

Keyword	Type	Description
		UPPER system switch.
SWITCH	PI	Enable or disable a system switch based on a value.
SWITCH	RF	Return an indication of the setting of a system switch.
$SYMBOL	ST	Determine the user symbol that is referenced by a pointer previously obtained with the SYMBOL.PTR real-valued function.
SYMBOL.PTR	RF	Determine the value of a pointer to a user symbol in eV+ memory.
$SYS.INFO	ST	This string function is intended to provide general system information. It also provides access to the ActiveVR log data.
TAS	RF	Return the current value of a real-valued variable and assign it a new value. The two actions are done indivisibly so that no other program task can modify the variable at the same time.
TASK	RF	Return information about a program execution task.
TIME	PI	Set the date and time.
TIME	RF	Return an integer value representing either the four-digit date or the time specified in the given string parameter.
$TIME	ST	Return a string value containing either the current system date and time or the specified date and time.
$TIME4	ST	Return a string value containing either the current system four-digit date and time or the specified four-digit date and time.
TIMER	PI	Set the specified system timer to the given time value.
TIMER	RF	Return the current time value of the specified system timer.
TOOL	PI	Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.
TOOL	TF	Return the value of the transformation specified in the last TOOL command or instruction.
TPS	RF	Return the number of ticks of the system clock that occur per second (Ticks Per Second).
TRANS	TF	Return a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.
TRANSB	TF	Return a transformation value represented by a 48-byte string.
$TRANSB	ST	Return a 48-byte string containing the binary representation of a transformation value.

Keyword	Type	Description
$TRUNCATE	ST	Return all characters in the input string until an ASCII NULL (or the end of the string) is encountered.
TYPE	PI	Display the information described by the output specifications on the system terminal. A blank line is output if no argument is provided.
$UNPACK	ST	Return a substring from an array of 128-character string variables.
UNTIL	PI	Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is nonzero.
UPPER	SW	Control whether or not the case of each character is ignored when string comparisons are performed.
VAL	RF	Return the real value represented by the characters in the input string.
VALUE	PI	Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.
VLOCATION	TF	Returns a Cartesian transform result of the execution of the specified vision sequence. The returned value is a transform result: x, y, z, yaw, pitch, roll. For details, see the *AdeptSight Reference Guide*.
VPARAMETER	PI	Sets the current value of a vision tool parameter. For details, see the *AdeptSight Reference Guide*. Can also be used with the Adept AnyFeeder. For details, see the *Adept AnyFeeder User's Guide*.
VPARAMETER	RF	Gets the current value of a vision tool parameter. For details, see the *AdeptSight Reference Guide*.
VRESULT	RF	Returns a specified result of a vision tool, or returns the status of a specified tool. For details, see the *AdeptSight Reference Guide*.
VRUN	PI	Initiates the execution of a vision sequence. For details, see the *AdeptSight Reference Guide*. Can also be used with the Adept AnyFeeder. For details, see the *Adept AnyFeeder User's Guide*.
VSTATE	RF	Returns the state of the execution of a sequence. For details, see the *AdeptSight Reference Guide*. Can also be used with the Adept AnyFeeder. For details, see the *Adept AnyFeeder User's Guide*.
VTIMEOUT	SP	Sets a timeout value so that an error message is returned if no

Keyword	Type	Description
		response is received following a vision command. For details, see the *AdeptSight Reference Guide*.
VWAITI	PI	Waits efficiently until the specified vision sequence reaches the state specified by the type parameter. For details, see the *AdeptSight Reference Guide*. Can also be used with the Adept AnyFeeder. For details, see the *Adept AnyFeeder User's Guide*.
WAIT	PI	Put the program into a wait loop for one system cycle. If a condition is specified, wait until the condition is TRUE.
WAIT.EVENT	PI	Suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed.
WHILE	PI	Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.
WINDOW	PI	Set the boundaries of the operating region of the specified belt variable for conveyor tracking.
WINDOW	RF	Return a value that indicates where the location described by the belt-relative transformation value is relative to the predefined boundaries of the working range on a moving conveyor belt.
WRITE	PI	Write a record to an open file, or to any I/O device. For a network device, write a string to an attached and open TCP connection.
XOR	O	Perform the *logical* exclusive-OR operation on two values.

Keyword Descriptions

The following topics are described in this chapter:

Descriptions of eV+ Keywords **42**
Documentation Conventions for Keywords **42**

Descriptions of eV+ Keywords

This chapter details the keywords in the eV+ programming language. The functional groups of programming keywords are:

- Program Instructions
- Functions
- System Parameters
- System Switches

This manual often refers to monitor commands. Monitor commands are part of the eV+ operating system. The eV+ operating system commands are detailed in the *eV+ Operating System Reference Guide*.

If your system is equipped with AdeptVision, additional program instructions, functions, switches, parameters, and monitor commands are detailed in the *AdeptVision Reference Guide*.

The keywords are presented in alphabetical order, with the description for each keyword starting on a new page. For details on what is included, see Documentation Conventions for Keywords.

Documentation Conventions for Keywords

The keyword type (function, program instruction, and so on) is shown at the top of the page.

Syntax

An abbreviated syntax is shown for some keywords. This is done when the abbreviated form is the most commonly used variation of the complete syntax.

This section presents the syntax of the keyword. The keyword is shown in uppercase, and the arguments are shown in lowercase. The keyword must be entered exactly as shown[1]. Parentheses must be placed exactly as shown. Required keywords, parameters, and marks such as equal signs and parentheses are shown in bold type; optional keywords, parameters, and marks are shown in regular type. In the example:

KEYWORD req.param1 = req.param2 OPT.KEYWORD opt.param

KEYWORD	must be entered exactly as shown,[1]
req.param1	must be replaced with a value, variable, or expression,

=	the equal sign must be entered,
req.param2	must be replaced with a value, variable, or expression,
OPT.KEYWORD	can be omitted but must be entered exactly as shown if used,
opt.param	may be replaced with a value, variable, or expression but assumes a default value if not used.

Function

This section gives a brief description of the keyword.

Usage Considerations

This section lists any restrictions on the keyword's use. If specific hardware or other options are required, they are listed here.

Parameters

The requirements for input and output parameters are explained in this section. If a parameter is optional, it is noted here. When an instruction line is entered, optional parameters do not have to be specified and the system will assume a default. Unspecified parameters at the end of an argument list can be ignored. Unspecified parameters in the middle of an argument list must be represented by commas. For example, the following keyword has four parameters-the first and third are used, and the second and fourth are left unspecified:

```
SAMPLE.INST var_1,,"test"
```

String and numeric input parameters can be constant values (3.32, part_1, etc.) or any legitimate variable names. The data type of the constant or variable must agree with that expected by the instruction. String variables must be preceded with a $. Precision-point variables must be preceded with a #. Belt variables must be preceded with a %. String constants must be enclosed in quotes. Real and integer constants can be used without modification. Note that some eV+ keywords cannot be used as variable names.

Details

This section describes the function of the keyword in detail.

Examples

Examples of correctly formed instruction lines are presented in this section.

Related Keywords

Additional keywords that are similar or are frequently used in conjunction with this instruction are listed here.

Any related keywords that are monitor commands are described in the *eV+ Operating System Reference Guide*.

[1]In the program editor, instructions can be abbreviated to a length that uniquely identifies the keyword. The SEE editor automatically expands the instruction to its full length.

ABORT program instruction

Syntax

ABORT task_num

Function

Terminate execution of an executing program task.

Usage Considerations

ABORT is ignored if no program is executing as the specified task.

ABORT does *not* force DETACH or FCLOSE operations on the disk or serial communication logical units. If the program has one or more files open and you decide not to resume execution of the program, use a KILL command to close all the files and detach the logical units.

Parameter

task_num	Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be terminated. The default task is 0.

Details

Terminates execution of the specified active executable program after completion of the step currently being executed. If the task is controlling a robot, robot motion terminates at the completion of the current motion. (Program execution can be resumed with the PROCEED command.)

Related Keywords

ABORT monitor command

CYCLE.END monitor command

CYCLE.END program instruction

ESTOP monitor command

ESTOP program instruction

EXECUTE program instruction

KILL monitor command

KILL program instruction

PANIC monitor command

PANIC program instruction

PROCEED monitor command

STATUS monitor command

STATUS real-valued function

ABOVE program instruction

Syntax

ABOVE

Function

Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.

Usage Considerations

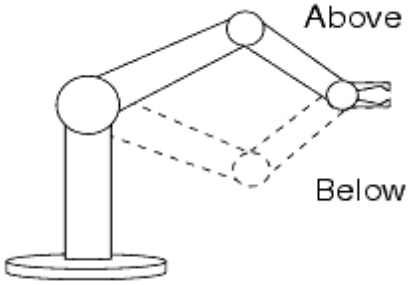
Configuration changes cannot be made during straight-line motions.

If the selected robot does not support an ABOVE configuration, this instruction is ignored by the robot (SCARA robots, for example, cannot have an ABOVE configuration).

The ABOVE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the ABOVE instruction causes an error.

The following figure shows the ABOVE and BELOW configurations.



ABOVE/BELOW

Related Keywords

BELOW program instruction

CONFIG real-valued function

SELECT program instruction

SELECT real-valued function

ABS real-valued function

Syntax

ABS (value)

Function

Return absolute value.

Parameter

value Real-valued expression.

Details

Returns the absolute value (magnitude) of the argument provided.

Examples

```
ABS(0.123)                  ;Returns 0.123
ABS(-5.462)                 ;Returns 5.462
ABS(1.3125E-2)              ;Returns 0.013125
belt.length = part.size/ABS(belt.scale)
```

ACCEL program instruction

Syntax

ACCEL (profile) acceleration, deceleration

Function

Set acceleration and deceleration for robot motions. Optionally, specify a defined acceleration profile.

Usage Considerations

The ACCEL instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the ACCEL instruction causes the error *Robot not attached to this program*.

Before an acceleration/deceleration profile can be used, it must be defined for the selected robot (profile 0 is always defined). (The robot configuration is edited using the Adept ACE software, see the *Adept ACE User's Guide*.)

Parameters

profile	Optional integer specifying the acceleration profile to use. Acceptable values are 0 to 8 (depending on the number of defined profiles). The default is the last specified profile (see Details for the number of the start-up profile). If a profile is specified that has not been defined, profile 0 is used.
acceleration	Optional real value, variable, or expression considered as a percentage of the maximum possible acceleration.
deceleration	Optional real value, variable, or expression considered as a percentage of the maximum possible deceleration. The value should normally be in the range of 1 to 100 (upper limits greater than 100 may be established by the robot manufacturer). If an out-of-range value is specified, the nearest extreme value will be used. If a parameter is omitted, its current setting remains in effect.

Details

If profile 0 is used, a square wave acceleration profile is generated at the beginning and end of the motion.

If a profile is specified, that profile is invoked for subsequent robot motions. Defined profiles set the maximum rate of change of the acceleration and deceleration. The values set with this instruction define the maximum acceleration and deceleration magnitudes that are achieved.

When the eV+ system is initialized, the profile, acceleration, and deceleration values are set to initial values, which can be defined by the ACE controller configuration tools. As delivered by Adept, the initially selected profile may be either 0 or 1 depending on the type of robot. The settings are not affected when program execution starts or stops, or when a ZERO command is processed.

Normally, the robot manufacturer sets the 100% acceleration and deceleration values to rates that can be achieved with typical payloads and robot link inertias. However, because the actual attainable accelerations vary greatly as a function of the end-effector, payload, and the initial and final locations of a motion, accelerations greater than 100% may be permitted for your robot. The limits for the maximum values are defined by the robot manufacturer and vary from one type of robot to the next. If you specify a higher acceleration than is permitted, the limit established by the robot manufacturer is utilized.

You can use the functions ACCEL(3) and ACCEL(4) to determine the maximum allowable acceleration and deceleration settings.

For a given motion, the maximum attainable acceleration may actually be less than what you have requested. This occurs when a profile with a nonzero acceleration ramp time is used and there is insufficient time to ramp up to the maximum acceleration. That is, for a given jerk, a specific time must elapse before the acceleration can be changed from zero to the specified maximum value. If the maximum acceleration cannot be achieved, the trapezoidal profile is reduced to a triangular shape. This occurs under two circumstances:

1. The motion is too short. In this case, the change in position is achieved before the maximum acceleration can be achieved.
2. The maximum motion speed is too low. In this case, the maximum speed is achieved before the maximum acceleration.

In both of these situations, raising the maximum acceleration and deceleration values does not affect the time for the motion.

Hint: If you increase the maximum acceleration and deceleration values but the motion time does not change, try the following: increase the program speed, switch to an acceleration profile that allows faster acceleration ramp times, or switch to acceleration profile 0, which specifies a square-wave acceleration profile.

NOTE: This type of acceleration limiting cannot occur with acceleration profile 0 because a square-wave acceleration instantaneously changes acceleration values without ramping.

Examples

Set the default acceleration time to 50% of normal and the deceleration time to 30% of normal:

```
ACCEL 50, 30
```

Change the deceleration time to 60% of normal; leave acceleration alone:

```
ACCEL ,60
```

Reduce the acceleration and deceleration to one half of their current settings:

```
ACCEL ACCEL(1)/2, ACCEL(2)/2
```

Invoke defined profile #2 and set the acceleration magnitude to 80% of the defined rate:

```
ACCEL (2) 80
```

Related Keywords

ACCEL real-valued function

DURATION program instruction

SCALE.ACCEL system switch

SELECT program instruction

SELECT real-valued function

SPEED monitor command

SPEED program instruction

ACCEL real-valued function

Syntax

ACCEL (select)

Function

Return the current setting for robot acceleration or deceleration setting or return the maximum allowable percentage limits defined in the robot configuration profile. (The robot configuration is edited using the Adept ACE software, see the *Adept ACE User's Guide*.)

Usage Considerations

The ACCEL function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the ACCEL function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select Real-valued expression, the result of which is rounded to an integer to select the value that is returned.

select	Value returned
0	Number of selected acceleration profile
1	Acceleration
2	Deceleration
3	Maximum allowable percentage acceleration
4	Maximum allowable percentage deceleration
5	Program speed below which acceleration and deceleration are scaled proportional to a program's speed setting when the SCALE.ACCEL system switch is enabled

Examples

```
ACCEL(1)      ;Return the current acceleration setting.
ACCEL(2)      ;Return the current deceleration setting.
```

Related Keywords

ACCEL program instruction

SCALE.ACCEL system switch

SELECT real-valued function

SELECT program instruction

ALIGN program instruction

Syntax

ALIGN

Function

Align the robot tool Z-axis with the nearest world axis.

Usage Considerations

The ALIGN instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the ALIGN instruction causes an error.

Details

Causes the tool to be rotated so that its Z-axis is aligned parallel to the nearest axis of the World coordinate system. This instruction is primarily useful for lining up the tool before a series of locations is taught. This is most easily done by using the monitor DO command.

Related Keywords

SELECT program instruction

SELECT real-valued function

ALWAYS keyword

Syntax

...ALWAYS

Function

Used with certain program instructions to specify a long-term effect.

Details

ALWAYS can be specified with any of the instructions listed below as related keywords. When ALWAYS is specified, the effect of the instruction continues until explicitly disabled. Otherwise, the effect of the instruction applies only to the next robot motion.

Examples

Permanently set the robot motion speed:

```
SPEED 50 ALWAYS
```

Permanently set loose-tolerance servo mode:

```
COARSE ALWAYS
```

Related Keywords

COARSE program instruction

CPOFF program instruction

CPON program instruction

DURATION program instruction

FINE program instruction

MULTIPLE program instruction

NONULL program instruction

NOOVERLAP program instruction

NULL program instruction

OVERLAP program instruction

SINGLE program instruction

SPEED program instruction

AND operator

Syntax

...value AND value...

Function

Perform the *logical* AND operation on two values.

Details

The AND operator operates on two values, resulting in their logical AND combination. For example, during the AND operation

```
c = a AND b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	->	FALSE
FALSE	TRUE	->	FALSE
TRUE	FALSE	->	FALSE
TRUE	TRUE	->	TRUE

The result is TRUE only if *both* of the two operand values are logically TRUE. To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Example

```
;The instructions following the IF will be executed if
;both "ready" is TRUE (nonzero) and "count" equals 1.
        IF ready AND (count == 1) THEN
```

Related Keywords

BAND operator

OR operator

XOR operator

ANY program instruction

Syntax

ANY

Function

Signal the beginning of an alternative group of instructions for the CASE structure.

Usage Considerations

The ANY instruction must be within a CASE structure.

Details

See the description of the CASE structure.

Related Keywords

CASE program instruction

VALUE program instruction

APPRO program instruction

Syntax

APPRO location, distance

APPROS location, distance

Function

Start a robot motion toward a location defined relative to specified location.

Usage Considerations

APPRO causes a joint-interpolated motion.

APPROS causes a straight-line motion, during which no changes in configuration are permitted.

The APPRO and APPROS instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions will cause an error.

Parameters

location	Transformation value that defines the basis for the final location.
distance	Real-valued expression that specifies the distance along the robot tool Z axis between the specified location and the actual desired destination. A positive distance sets the tool back (negative tool-Z) from the specified location; a negative distance offsets the tool forward (positive tool-Z).

Details

These instructions initiate a robot motion to the orientation described by the given location value. The position of the destination location is offset from the given location by the distance given, measured along the tool Z axis.

Examples

```
APPRO place,offset
```

Moves the tool, by joint-interpolated motion, to a location offset millimeters from that defined by the transformation place. The offset is along the resultant Z axis of the tool.

```
APPROS place,-50
```

Moves the tool along a straight line to a location 50 millimeters from that defined by the transformation place, with the offset along the resultant Z axis of the tool to a location **beyond** the location place.

Related Keywords

DEPART program instruction

DEPARTS program instruction

MOVE program instruction

MOVES program instruction

APPRO program instruction

Syntax

APPRO location, distance

APPROS location, distance

Function

Start a robot motion toward a location defined relative to specified location.

Usage Considerations

APPRO causes a joint-interpolated motion.

APPROS causes a straight-line motion, during which no changes in configuration are permitted.

The APPRO and APPROS instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions will cause an error.

Parameters

location	Transformation value that defines the basis for the final location.
distance	Real-valued expression that specifies the distance along the robot tool Z axis between the specified location and the actual desired destination. A positive distance sets the tool back (negative tool-Z) from the specified location; a negative distance offsets the tool forward (positive tool-Z).

Details

These instructions initiate a robot motion to the orientation described by the given location value. The position of the destination location is offset from the given location by the distance given, measured along the tool Z axis.

Examples

```
APPRO place,offset
```

Moves the tool, by joint-interpolated motion, to a location offset millimeters from that defined by the transformation place. The offset is along the resultant Z axis of the tool.

```
APPROS place,-50
```

Moves the tool along a straight line to a location 50 millimeters from that defined by the transformation place, with the offset along the resultant Z axis of the tool to a location **beyond** the location place.

Related Keywords

DEPART program instruction

DEPARTS program instruction

MOVE program instruction

MOVES program instruction

ASC real-valued function

Syntax

ASC (string, index**)**

Function

Return an ASCII character value from within a string.

Parameters

string	String expression from which the character is to be picked. If the string is empty, the function returns the value -1.
index	Optional real-valued expression defining the character position of interest. The first character of the string is selected if the index is omitted or has a value of 0 or 1. If the value of the index is negative, or greater than the length of the string, the function returns the value -1.

Details

The ASCII value of the selected character is returned as a real value.

Examples

```
;Returns the ASCII value of the letter "a".

    ASC("sample", 2)

;Returns the ASCII value of the first character of the
;string contained in the variable $name.

    ASC($name)

;Uses the value of the real variable "i" as an index to
;the character of interest in the string contained in the
;variable "$system".

    ASC($system, i)
```

Related Keywords

$CHR string function

VAL real-valued function

ATAN2 real-valued function

Syntax

ATAN2 (value_1, value_2)

Function

Return the size of the angle (in degrees) that has its trigonometric tangent equal to value_1/value_2.

Usage Considerations

The returned value is zero if both parameter values are zero.

Parameters

value_1 Real-valued expression.

value_2 Real-valued expression.

Examples

```
ATAN2(0.123,0.251)          ;Returns  26.1067
ATAN2(-5.462,47.2)          ;Returns  -6.600926
ATAN2(1.3125E+2,-1.3)       ;Returns  -90.56748
slope = ATAN2(rise, run)
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

ATTACH program instruction

Syntax

ATTACH (lun, mode) $device

Function

Make a device available for use by the application program.

Usage Considerations

The robot is automatically attached when the EXECUTE monitor command or program instruction is processed for task 0 (except when the DRY.RUN system switch is enabled). All the other logical units are automatically detached when program execution begins.

If the system terminal or the pendant was attached when a program stopped executing, it is automatically reattached if execution of the program is resumed with the PROCEED, RETRY, SSTEP, or XSTEP commands.

Parameters

lun — The logical unit number to associate with the attached device. The interpretation of this parameter depends on the value of the mode parameter, as follows:

If bit 3 of the mode parameter is 0, this parameter is optional (defaulting to 0, to attach the robot); and it can be a real value, variable, or expression (interpreted as an integer) in the range 0 to 24 that specifies the logical unit to be attached. See the Details section for the default association of logical units with devices. If the logical unit specified is not 0, you can use the $device parameter to override the default device for the logical unit.

If bit 3 of the mode parameter is 1, this parameter is required and must be a real variable. In this case, the eV+ system attaches the device specified by the $device parameter and automatically assigns a logical unit number to this parameter. If all the logical units are in use, the parameter is set to -1 . eV+ assigns a value to the lun parameter even if the ATTACH request fails.

mode — Optional real value, variable, or expression (interpreted as a bit field) that defines how the ATTACH request is to be processed. The value specified is interpreted as a sequence of bit flags as detailed below. All the bits are assumed to be clear if no value is specified.

Bit 1 (mask value = 1) - (LSB) Queue (0) versus Fail (1)

This bit controls how the device driver responds to the attach request from the control program task. (The device driver is an internal system task that is separate from the control program task.) For most applications, this bit should be set.

If this bit is clear, and the device is already attached by another control program task, the driver queues this attach request and signals the control program that the attach is not complete. The attachment will complete when the device becomes available.

If this bit is set, and the device is already attached by another control program task, the device driver immediately signals that the attach request has failed.

The function IOSTAT(lun) can be used to determine the success or failure of the attachment. A positive value from IOSTAT indicates successful completion; zero indicates the attachment has not completed; a negative value indicates completion with an error.

Bit 2 (mask value = 2) - Wait (0) versus No-wait (1)

This bit controls whether or not the control program task waits for a response from the device driver. For most applications, this bit should not be set.

If this bit is clear, program execution waits for the device driver to signal the result of the attach request.

If this bit is set, program execution does not wait for the result of the attach request. The program must then use the function IOSTAT(lun) to determine if the attachment has succeeded (see earlier text). If the program attempts to READ from or WRITE to the logical unit while the attachment is pending, program execution then waits for the attachment to complete.

Bit 3 (mask value = 4) - Specify LUN (0) versus Have LUN Assigned (1)

This bit determines how the lun parameter is processed.

If this bit is clear, the device corresponding to the value of lun is attached. That is, the value of the lun parameter specifies the device that is to be attached (according to the table in the Details section) except when a different device is specified with the $device parameter.

If this bit is set, the device to be attached is specified by the $device parameter (which should not be omitted). In this case, a logical unit is automatically selected, and the value of the lun parameter is *set* by the ATTACH instruction. eV+ assigns a value to lun even if the ATTACH request fails. (This mode cannot be used to attach the robot or

pendant.)

$device — Optional string constant, variable, or expression that identifies the device to be attached. If bit 3 of the mode parameter is 0, this parameter is used to override the default device associated with the value of the lun parameter (except that logical unit 0 is always the robot).

The acceptable device names are shown in the following table.

Acceptable Device Names to Be Attached

Device	Meaning
DISK	Physical drive in the Adept controller (disk orSecure Digitalcard)
DEVICENET	Access devices connected to DeviceNet
MONITOR	The current monitor window or operator's terminal
SERIAL:n	Global serial line (n = 0, 1, 2 or 3). For the Adept SmartController EX, - SERIAL:0 is RS232/TERM, - SERIAL:1 is RS232-1, - SERIAL:2 is RS232-2, - SERIAL:3 is RS-422/485
SYSTEM	Disk device, currently set with the CD or DEFAULT command
TCP	TCP protocol device driver
TFTP	Access the TFTP server to read files
UDP	UDP protocol device driver

Details

The robot must remain attached by a robot control program for the program to command motion of the robot. When the robot is detached (see the DETACH instruction), however, you can use the manual control pendant to move the robot under directions from the application program. This is useful, for example, for application setup sequences. (The belt and vision calibration programs provided by Adept use this technique.)

Program task 0 automatically attaches robot #1 when that task begins execution. A robot control program executed by any of the other program tasks must explicitly attach the robot.

Any task can attach to any robot, provided that the robot is not already attached by a different task. The robot that is attached by an ATTACH instruction is the one that was last specified by a SELECT instruction executed by the current task (see the SELECT instruction). If no SELECT instruction has been executed, then robot #1 is attached. The SELECT instruction can be used to select a different robot only if no robot is currently attached to the task.

To successfully attach the robot, the system must be in COMP mode. Otherwise (for mode bit 1 = 0), program execution is suspended (without notice) until the system is placed in COMP mode. This situation can be avoided in two ways: (1) use the STATE function to determine if the system is in COMP mode before executing an ATTACH instruction, (2) set bit 1 in the mode value, and use the IOSTAT function to determine the success of the ATTACH instruction.

When the system terminal (logical unit 4) is attached, all keyboard input will be buffered for input requests by the program.

> **NOTE:** When the system terminal is attached, a user is not able to type ABORT to terminate program execution. The program must provide a means for fielding a termination request, or you must use the pendant or emergency stop switch to stop program execution.

When a DISK device is attached, it allows a program to read and write data from and to files. DISK refers to the Secure Digital (SD) card. One of the FOPEN instructions must be used to specify which file to access. WRITE and READ instructions can then be used to transfer information to and from the file. Also, FCMND instructions can be used to send commands to the file system.

When a TFTP device is attached, it allows a program to read a file from an TFTP server.

When a SERIAL:n serial communication line is attached, it can be used to send and receive information to and from another system. As with disk I/O, WRITE and READ instructions are used for the information transfer. For the details on physical connectors and corresponding eV+ designations, see: Table 3-5. Serial Connectors and eV+ Designations in the *Adept SmartController User's Guide*.

When **mode** bit 3 = 0 and the **$device** parameter is omitted, the logical unit number implicitly specifies the corresponding default device from the following table.

Default Device Numbers Supplied by the LUN

Number	Device
0	Robot (default when lun is omitted)

Number	Device
2	System terminal
3	System terminal
4	System terminal
5	Disk
6	Disk
7	Disk
8	Disk
9	No default device
10	Serial communication line (SERIAL:0)
11	Serial communication line (SERIAL:1)
12	Serial communication line (SERIAL:2)
13	Serial communication line (SERIAL:3)
14	No default device
15	No default device
16	No default device
17	Disk
18	Disk
19	Disk

Number	Device
25 - 31	No default devices
NOTE: 1. There are currently no default LUNs assigned for serial communication lines Local #3 or Local #4.	

Examples

- Take over control of the robot:

```
ATTACH
```

- Connect to global serial line 1; wait for it to become available if another task has it attached; return the assigned logical unit number in lun:

```
ATTACH (lun, 4) "serial:1"
```

- The next instruction is similar to the previous one, but this one requires use of the IOSTAT function to determine if another task has the serial line attached:

```
ATTACH (lun, 5) "serial:1"
```

- Attach to the TCP device driver with automatic allocation of a logical unit number:

```
ATTACH (lun, 4) "TCP"
```

Related Keywords

DETACH program instruction

FSET program instruction

IOSTAT real-valued function

SELECT program instruction

AUTO program instruction

Syntax

AUTO type **variable,** ..., variable

Function

Declare temporary variables that are automatically created on the program stack when the program is entered.

Usage Considerations

AUTOmatic variables have an undetermined value when a program is first entered (but they are *not* necessarily undefined), and they have no value after the program exits.

AUTO statements must appear before any executable instruction in the program-only the .PROGRAM statement, comments, blank lines, GLOBAL and LOCAL statements, and other AUTO statements may precede this instruction.

If a variable is listed in an AUTO statement, any global variables with the same name cannot be accessed directly by the program.

The values of AUTOmatic variables are not saved by the STORE or restored by the LOAD monitor commands.

Parameters

type	Optional keyword REAL, DOUBLE, or LOC, indicating that all the variables in this statement are to be single precision, double precision, or location variables. (A location can be a transformation, precision point, or belt variable.) If this keyword is omitted, the type of each variable is determined by its use within the program. An error is generated if the type cannot be determined from usage.
variable	Name of a variable of any data type available with eV+ (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array. If the **type** parameter is specified (see below), all the variables must match that type. Array variables must have their indexes specified explicitly, indicating the highest valid index for the array.

Details

This instruction is used to declare variables to be defined only within the current program. That is, an AUTOmatic variable can be referenced only by the specific calling instance of a program. Also, the names of AUTOmatic variables can be selected without regard for the names of variables defined in any other programs.

AUTOmatic variables are allocated each time the program is called, and their values are not preserved between successive subroutine calls. These values can be displayed via monitor commands only when the program task is inactive but is on an execution stack. When a program is first entered, automatic variables have arbitrary, undetermined values (and they are *not* necessarily undefined). AUTOmatic variables are lost when the program exits.

Unlike a LOCAL variable, a separate copy of an AUTOmatic variable is created *each time* a program is called, even if it is called simultaneously by several different program tasks, or called recursively by a single task. If a program that uses LOCAL or global variables is called by several different program tasks, or recursively by a single task, the values of those variables can be modified by the different program instances and can cause very strange program errors. Therefore, AUTOmatic variables should be used for all temporary local variables to minimize the chance of such errors.

Variables can be defined as GLOBAL, AUTOmatic, or LOCAL. An attempt to define AUTOmatic, GLOBAL, or LOCAL variables with the same name will result in the error message *Attempt to redefine variable class*.

Variables can be defined only once within the same context (AUTOmatic, LOCAL, or GLOBAL). Attempting to define a variable more than once (that is, with a different type) will yield the error message

```
*Attempt to redefine variable type*
```

AUTOmatic array variables must have the size of each dimension specified in the AUTO statement. Each index specified must represent the last element to be referenced in that dimension. The first element allocated always has index value zero. For example, the statement

```
AUTO LOC points[3,5]
```

allocates a transformation array with 24 elements. The left-hand index ranges from 0 to 3, and the right-hand index ranges from 0 to 5.

The storage space for AUTOmatic variables is allocated on the program execution stack. If the stack is too small for the number of AUTOmatic variables declared, the task execution will stop with the error message

```
*Too many subroutine calls*
```

If this happens, use the STATUS monitor command to determine how much additional stack space is required. Then, use the STACK monitor command to increase the stack size and then issue the RETRY monitor command to continue program execution.

AUTOmatic variables cannot be deleted with the DELETE_ commands.

AUTOmatic variables can be referenced with monitor commands such as BPT, DELETE_, DO, HERE, LIST_, POINT, TEACH, TOOL, and WATCH by using the optional context specifier @. The general syntax is:

```
command @task:program command_arguments
```

Examples

- Declare the variables **loc.a**, **$ans**, and **i** to be AUTOmatic in the current program (the variable types for **loc.a** and **i** must be clear from their use in the program):

  ```
  AUTO loc.a, $ans, i
  ```

- Declare the variables **i**, **j**, and **tmp[ ]** to be AUTOmatic, real variables in the current program (array elements **tmp[0]** through **tmp[10]** are defined):

  ```
  AUTO REAL i, j, tmp[10]
  ```

- Declare the variable **loc** to be an AUTOmatic variable in the current program. The variable type of **loc** must be determined by its use in the program. Note that since LOC appears by itself, it is *not* interpreted as the type-specifying keyword.)

  ```
  AUTO loc
  ```

Related Keywords

GLOBAL program instruction

LOCAL program instruction

STACK monitor command

AUTO.POWER.OFF system switch

Syntax

AUTO.POWER.OFF

Function

Control whether or not eV+ disables high power when certain motion errors occur.

Usage Considerations

This switch has effect during automatic mode but not during manual mode. It is especially useful in reducing operator intervention during common nulling-timeout and envelope errors.

Details

Because the HIGH POWER ON/OFF high power on/off button cannot be used by itself to enable high power as in earlier versions of eV+, Adept has sought to reduce the number of instances that high power is disabled during normal program execution. Making this improvement allows programs to continue to recover automatically from errors without manual intervention, that is, without requiring you to press the HIGH POWER ON/OFF button. This system switch cancels the effect of this change. By default this switch is disabled. Enabling it restores functionality as it was in eV+ version 11.x and earlier.

Adept reviewed all automatic-mode errors that disabled high power in eV+ version 12.0 and determined which can be changed simply to decelerate the robot and generate an error without compromising the safe operation of the system. Examples of particular importance are errors such as nulling-timeout and envelope errors that often occur during the normal operation of the system. In some cases, Adept has modified internal software to ensure the continued safe operation of your system.

The setting of this switch has no effect during manual mode.

If this switch is enabled, eV+ disables high power for all motion related errors including nulling-timeout and soft envelope errors.

Example

The following program segment instructs eV+ to disable high power when any motion error occurs:

```
ENABLE AUTO.POWER.OFF            ;Disable high power when any
                                 ;motion error occurs
```

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

BAND operator

Syntax

...value BAND value...

Function

Perform the *binary* AND operation on two values.

Usage Considerations

The BAND operation is meaningful only when performed on integer values. Only the integer parts of real values are used. Any fractional parts are ignored.

Details

The BAND operator can be used to perform a binary AND operation on two values on a bit-by-bit basis, resulting in a real value.

Specifically, the BAND operation consists of the following steps:

1. Convert the operands to sign-extended 32-bit integers, truncating any fractional part.
2. Perform a binary AND operation (see below).
3. Convert the result back to a real value.

During the binary AND operation,

```
c = a BAND b
```

the bits in the resultant C are determined by comparing the corresponding bits in the operands A and B as indicated in the following table.

For each bit in:			
a	**b**		**c**
0	0	->	0
0	1	->	0
1	0	->	0
1	1	->	1

That is, a bit in the result will be 1 if the corresponding bit in both of the operands is 1.

To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Examples

Consider the following (binary values are shown to make the operation more evident):

^B101000 BAND ^B100001 yields ^B100000 (32)

Note that a very different result is obtained with the logical AND operation:

^B101000 AND ^B100001 yields -1 (TRUE)

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

AND operator

BOR operator

BXOR operator

BASE program instruction

Syntax

BASE X_shift, Y_shift, Z_shift, Z_rotation

Function

Translate and rotate the World reference frame relative to the robot.

Usage Considerations

The BASE program instruction causes a BREAK in continuous-path motion.

The BASE monitor command applies to the robot selected by the eV+ monitor (with the SELECT command). The command can be used while programs are executing. However, an error will result if the robot is attached by any executing program.

The BASE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, use of the BASE command or instruction will cause an error.

Parameters

X_shift	Optional real-valued expression describing the X component (in the normal World coordinate system) of the origin point for the new coordinate system. (Zero is assumed if no value is provided.)
Y_shift	Similar to X_shift, but for the Y direction.
Z_shift	Similar to X_shift, but for the Z direction.
Z_rotation	Similar to X_shift, but for a rotation about the Z axis.

Details

When the eV+ system is initialized, the origin of the reference frame of the robot is assumed to be fixed in space such that the X-Y plane is at the robot mounting surface, the X axis is in the direction defined by joint 1 equal to zero, and the Z axis coincides with the joint-1 axis.

The BASE instruction offsets and rotates the reference frame as specified above. This is useful if the robot is moved after locations have been defined for an application.

If, after robot locations have been defined by transformations relative to the robot reference frame, the robot is moved relative to those locations-to a point translated by dX, dY, dZ and rotated by Z rotation degrees about the Z axis—a BASE command or instruction can be used to compensate so that motions to the previously defined locations will still be as desired.

Another convenient use for the BASE command or instruction is to realign the X and Y coordinate axes so that SHIFT functions cause displacements in desired, nonstandard directions.

> **NOTE:** The BASE instruction has no effect on locations defined as precision points. The arguments for the BASE instruction describe the displacement of the robot relative to its normal location.
>
> The BASE function can be used with the LISTL command to display the current BASE setting.

Examples

```
BASE xbase,, -50.5, 30
```

Redefines the World reference frame because the robot has been shifted xbase millimeters in the positive X direction and 50.5 millimeters in the negative Z direction, and has been rotated 30 degrees about the Z axis.

```
BASE 100,, -50
```

Redefines the World reference frame to effectively shift all locations 100 millimeters in the negative X direction and 50 millimeters in the positive Z direction from their nominal location. Note that the arguments for this instruction describe movement of the robot reference frame relative to the robot, and thus have an opposite effect on locations relative to the robot.

Related Keywords

BASE transformation function

SELECT monitor command

SELECT program instruction

SELECT real-valued function

BASE transformation function

Syntax

BASE

Function

Return the transformation value that represents the translation and rotation set by the last BASE command or instruction.

Usage Considerations

The BASE function returns information for the robot selected by the task executing the function.

The command LISTL BASE can be used to display the current base setting.

If the eV+ system is not configured to control a robot, use of the BASE function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Related Keywords

BASE monitor command

BASE program instruction

SELECT program instruction

SELECT real-valued function

BCD real-valued function

Syntax

BCD (value)

Function

Convert a real value to Binary Coded Decimal (BCD) format.

Usage Considerations

The BCD function is most useful when used in conjunction with the BITS command, instruction, and function (see below).

Parameter

value	Real-valued expression defining the value to be converted.

Details

The BCD function converts an integer value in the range 0 to 9999 into its BCD representation. This can be used to set a BCD value on a set of external output signals.

Example

If you want to use digital signals 4 to 8 to output a BCD digit: The instruction

```
BITS 4,4 = BCD(digit)
```

converts the value of the real variable digit to BCD and impresses it on external output signals 4-8.

Related Keyword

DCB real-valued function

BELOW program instruction

Syntax

BELOW

Function

Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.

Usage Considerations

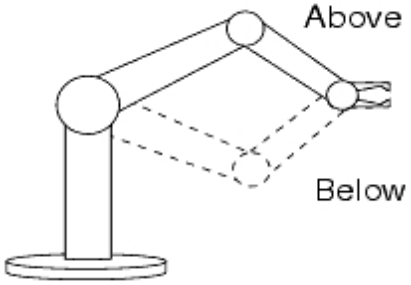
Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a below configuration, this instruction is ignored by the robot. (SCARA robots, for example, cannot be in an ABOVE/BELOW configuration.)

The BELOW instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the BELOW instruction will cause an error.

The following figure shows the ABOVE and BELOW configurations.



ABOVE/BELOW

Related Keywords

ABOVE program instruction

CONFIG real-valued function

SELECT program instruction

SELECT real-valued function

BELT real-valued function

Syntax

BELT (%belt_var, mode**)**

Function

Return information about a conveyor belt being tracked with the conveyor tracking feature.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The BELT system switch must be enabled before this function can be used.

The SETBELT instruction is generally used in conjunction with the BELT real-valued function to set the effective belt position to zero. This must be done each time the robot will perform a sequence of motions relative to the belt, and must be done shortly before the first motion of such a sequence.

WARNING: It is important to execute SETBELT each time the robot is going to track the belt, to make sure the difference between the current belt position (as returned by the BELT function) and the belt position of the specified belt variable does not exceed 8,388,607 (^H7FFFFF) during active belt tracking. Unpredictable robot motion may result if the difference does exceed this value while tracking the belt.

Parameters

%belt_var	The name of the belt variable used to reference the conveyor belt. As with all belt variables, the name must begin with a percent symbol (%).
mode	Control value that determines the information that will be returned. If the mode is omitted or its value is equal to zero, the BELT function returns the encoder reading in encoder counts of the belt specified by the belt variable. The value returned by this function is limited to an absolute value of 8,388,607 and will roll over to -8,388,608 after. If the value is equal to -1, the BELT function returns the last latched encoder position in encoder counts of the belt specified by the belt variable. This value equivalent to the value returned by DEVICE(0, enc, stt, 4) except it is not bounded to 8,388,607.

If the value of the expression is greater than zero, the encoder velocity is returned in units of encoder counts per eV+ cycle (16 ms).

Examples

Set the point of interest on the referenced conveyor to be that corresponding to the current reading of the belt encoder:

```
SETBELT %main.belt = BELT(%main.belt)
```

Save the current speed of the belt associated with the belt variable %b:

```
belt.speed = BELT(%b, 1)
```

Related Keywords

BELT system switch

SETBELT program instruction

BELT system switch

Syntax

...BELT

Function

Control the function of the conveyor tracking features of the eV+ system.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

If the eV+ system is not configured to control a robot, an attempt to enable the BELT system switch will cause an error. (The DEVICE real-valued function and the SETDEVICE program instruction must be used to access external encoders from a nonrobot system. For more information, see the section External Encoder Device in the *eV+ Language User's Guide*.)

Details

This switch must be enabled before any of the special conveyor tracking instructions can be executed. When BELT is disabled, the conveyor tracking software has a minimal impact on the overall performance of the system.

When the BELT switch is enabled, error checking is initiated for the encoders associated with any belt variables that are defined. The switch is *disabled* when the eV+ system is initialized.

Related Keywords

BELT real-valued function

BELT.MODE system parameter

BSTATUS real-valued function

DEFBELT program instruction

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SETBELT program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

WINDOW program instruction

WINDOW real-valued function

BELT.MODE system parameter

Syntax

...BELT.MODE

Function

Set characteristics of the conveyor tracking feature of the eV+ system.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The current value of the BELT.MODE parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the BELT.MODE parameter can be modified only with the PARAMETER monitor command or program instruction.

Details

This parameter is interpreted as a bit-flag word. The initial setting of this parameter is 0. That is, all the bits are zero. Bits can be set by assigning the value resulting from adding together the desired bit mask values (see the example below).

The bit flags have the following interpretations:

Bit 1 (LSB)	Upstream/downstream definition (mask value = 1) When this bit is set to one, the instantaneous direction of travel of the belt is used to define upstream and downstream for the window testing routines (both in the internal motion planner and the WINDOW real-valued function). When this bit is set to zero, going from upstream to downstream always corresponds to traveling in the direction of the positive X axis of the nominal transformation.
Bit 2	Stopped-belt processing (mask value = 2) When this bit is set to one, a program error will be generated during motion planning if the destination is outside of the belt window and the belt is stopped. When this bit is set to zero, if the belt is stopped during motion planning, the direction of the positive X axis of the nominal transformation is used to define the downstream direction. The normal window-error criteria are then applied (see below).

Bit 3 Window error definition (mask value = 4)

When this bit is set to one, destination locations that are downstream or upstream of the belt window cause motion instructions to fail during planning.

When this bit is set to zero, upstream window violations cause planning to wait until the location comes into the window. Destination locations that are downstream of the belt window cause window errors.

Bit 4 Effect of window errors (mask value = 8)

When this bit is set to one, motion instructions that fail during planning due to a window error are ignored (skipped) and program execution continues as usual. When this option is selected, each belt-relative motion instruction should be followed by an explicit test for planning errors using the BSTATUS function.

When this bit is zero, window errors during motion planning generate a program step execution error, which either halts program execution or triggers the REACTE routine.

Regardless of the setting of this bit, window errors that occur while the robot is actually tracking the belt cause the program specified in the latest WINDOW instruction to be executed. If no such program has been specified, program execution is halted.

Example

Set the parameter to have bits 1 and 3 set to one (mask values 1 + 4):

```
PARAMETER BELT.MODE = 5
```

Related Keywords

BELT system switch

BELT real-valued function

BSTATUS real-valued function

PARAMETER monitor command

PARAMETER program instruction

PARAMETER real-valued function

WINDOW program instruction

WINDOW real-valued function

BITS program instruction

Syntax

BITS first_sig, num_sigs **= value**

Function

Set or clear a group of digital signals based on a value.

Usage Considerations

Both external digital output signals and internal software signals can be referenced. Input signals must not be referenced. (Input signals are displayed by the monitor command IO 1.)

No more than 32 signals can be set at one time.

Any group of up to 32 signals can be set, provided that all the signals in the group are configured for use by the system.

Parameters

first_sig	Real-valued expression defining the lowest-numbered signal to be affected.
num_sigs	Optional real-valued expression specifying the number of signals to be affected. A value of 1 is assumed if none is specified. The maximum valid value is 32 .
value	Real-valued expression defining the value to be set on the specified signals. If the binary representation of the value has more bits than num_sigs, only the lowest num_sigs signals will be affected.

Details

Sets or clears one or more external output signals or internal software signals based on the value to the right of the equal sign. The effect of this instruction is to round **value** to an integer, and then set or clear a number of signals based on the individual bits of the binary representation of the integer.

All eV+ digital output instructions do not wait for a eV+ cycle, they turned on outputs immediately. However, digital inputs are checked every 2 milliseconds by the eV+ operating system.

Examples

Set external output signals 1-8 (8 bits) to the binary representation of the current monitor speed setting:

```
BITS 1,8 = SPEED(1)
```

If the monitor speed were currently set to 50% (0011 0010 binary), then signals 1-8 are set as follows after this instruction:

signal 1 -> 0 (off) signal 5 -> 1 (on)

signal 2 -> 1 (on) signal 6 -> 1 (on)

signal 3 -> 0 (off) signal 7 -> 0 (off)

signal 4 -> 0 (off) signal 8 -> 0 (off)

Set external output signals 5-9 (4 bits) to the binary representation of the BCD digit 7:

```
BITS 5,4 = BCD(7)
```

Set external output signals 1-8 (8 bits) to the binary representation of the constant 255, which is 11111111 (binary). Thus, signals 1-8 will all be turned on:

```
BITS 1,8 = 255
```

Related Keywords

BITS real-valued function

IO monitor command

RESET monitor command

SIG real-valued function

SIG.INS real-valued function

SIGNAL monitor command

SIGNAL program instruction

BITS real-valued function

Syntax

BITS (first_sig, num_sigs**)**

Function

Read multiple digital signals and return the value corresponding to the binary bit pattern present on the signals.

Usage Considerations

External digital input or output signals, or internal software signals can be referenced.

A maximum of 32 signals can be read at one time.

Any group of up to 32 signals can be read, provided that all the signals in the group are configured for use by the system.

Parameters

first_sig	Real-valued expression defining the lowest-numbered signal to be read.
num_sigs	Optional real-valued expression specifying the number of signals to be affected. A value of 1 is assumed if none is specified. The maximum valid value is 32 .

Details

This function returns a value that corresponds to the binary bit pattern present on 1 to 32 digital signals.

The binary representation of the value returned by the function has its least-significant bit determined by signal numbered **first_sig**, and its higher-order bits determined by the next num_sigs -1 signals.

Example

Assume that the following input signal states are present:

Signal:	1008	1007	**1006**	**1005**	**1004**	**1003**	1002	1001
State:	1	1	**0**	**1**	**0**	**1**	1	0

The program step:

```
x = BITS(1003, 4)
```

will yield a value of 5 for x since the four signals starting at 1003 (that is, signals 1003 through 1006) can be interpreted as a binary representation of that value.

Related Keywords

BITS monitor command

BITS program instruction

IO monitor command

RESET monitor command

SIG real-valued function

SIG.INS real-valued function

SIGNAL monitor command

SIGNAL program instruction

BMASK real-valued function

Syntax

BMASK (bit, bit**, ...,** bit**)**

Function

Create a bit mask by setting individual bits.

Parameter

bit	Integer value from 1 to 32 specifying a bit to turn on. The least-significant bit is number 1.

Details

This instruction creates a bit mask by turning on (bit = 1) the specified bits and leaving all other bits off (bit = 0).

Bit 32 is the sign bit and yields a negative number when set.

Examples

Create the bit mask ^B10001:

```
bm = BMASK(1, 5)
```

Attach to a disk LUN with mode bit 2 turned on:

```
mode = BMASK(2)
ATTACH (lun, mode) "DISK"
```

BOR operator

Syntax

... value BOR value ...

Function

Perform the binary OR operation on two values.

Usage Considerations

The BOR operation is meaningful only when performed on integer values. Only the integer parts of real values are used. Any fractional parts are ignored.

Details

The BOR operator can be used to perform a binary OR operation on two values on a bit-by-bit basis, resulting in a real value.

Specifically, the BOR operation consists of the following steps:

1. Convert the operands to sign-extended 32-bit integers, truncating any fractional part.
2. Perform a binary OR operation (see below).
3. Convert the result back to a real value.

During the binary OR operation,

```
c = a BOR b
```

the bits in the resultant C statement are determined by comparing the corresponding bits in the operands A and B, as indicated in the following table.

For each bit in:			
a	**b**		**c**
0	0	->	0
0	1	->	1
1	0	->	1
1	1	->	1

That is, a bit in the result will be 1 if the corresponding bit in either of the operands is 1.

To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Examples

Consider the following (binary values are shown only to make the operation more evident):

^B101000 BOR ^B100001 yields ^B101001 (41)

Note that a very different result is obtained with the logical OR operation:

^B101000 OR ^B100001 yields -1 (TRUE)

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

BAND operator

BXOR operator

OR operator

BRAKE program instruction

Syntax

BRAKE

Function

Abort the current robot motion.

Usage Considerations

The BRAKE instruction can be executed by any program task, including a task that is not actively controlling the robot.

This instruction does *not* cause a BREAK to occur (see Details below).

If more than one robot is connected to the controller, this instruction applies to the robot currently selected (see the SELECT instruction).

If the eV+ system is not configured to control a robot, the BRAKE instruction will not generate an error due to the absence of a robot.

Details

BRAKE causes the current robot motion to be aborted immediately. In response to this instruction, the robot will decelerate to a stop and then (without waiting for position errors to null) begin the next motion.

> **NOTE:** Program execution is not suspended until the robot motion stops.

Example

The following program segment initiates a robot motion and simultaneously tests for a condition to be met. If the condition is met, the motion is stopped with a BRAKE instruction. Otherwise, the motion is completed normally.

```
MOVES step[1]             ;Initiate motion to next location
DO                        ;Loop continuously...
    IF SIG(1023) THEN     ;If input signal 1023 becomes set,
        BRAKE             ;stop the motion immediately
        EXIT              ;and continue elsewhere
    END
UNTIL STATE(2) == 2       ;...until location reached
MOVES step[2]             ;Move to next location
```

Related Keyword

BREAK program instruction

BREAK program instruction

Syntax

BREAK

Function

Suspend program execution until the current motion completes.

Usage Considerations

The BREAK instruction is only used to wait for motion by the robot attached to the current task.

If the eV+ system is not configured to control a robot, executing the BREAK instruction will cause an error.

Details

This instruction has two effects:

1. Program execution is suspended until the robot reaches its current destination.

   > **NOTE:** BREAK cannot be used to have one task wait until a motion is completed by another task.

2. The continuous-path transition between the current motion and that commanded by the next motion instruction is broken. That is, the two motions are prevented from being merged into a single continuous path.

The BREAK instruction causes continuous-path processing to terminate by blocking eV+ program execution until the motion ends. CPOFF causes the trajectory generator to terminate continuous path without affecting the forward processing of the eV+ program.

Related Keywords

BRAKE program instruction

CP system switch

SELECT program instruction

SELECT real-valued function

BSTATUS real-valued function

Syntax

BSTATUS

Function

Return information about the status of the conveyor tracking system.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The BSTATUS function returns information for the robot selected by the task executing the function.

The word "bstatus" cannot be used as a program name or variable name.

Details

This function is normally used when BELT.MODE bit 4 is set.

This function returns a value that is equivalent to the binary value represented by a set of bit flags, which indicate the following conditions of the conveyor tracking software:

Bit 1 (LSB)	Tracking belt (mask value = 1) When this bit is set, the robot is currently tracking a belt.
Bit 2	Destination upstream (mask value = 2) When this bit is set, the destination location was found to be upstream of the belt window during the planning of the last motion.
Bit 3	Destination downstream (mask value = 4) When this bit is set, the destination location was found to be downstream of the belt window during the planning of the last motion.
Bit 4	Window violation (mask value = 8) When this bit is set, a window violation occurred while the robot was tracking a belt during the last belt-relative motion. (This flag is cleared at the start of each belt-relative motion.)

Related Keywords

BELT real-valued function

BELT system switch

BELT.MODE system parameter

DEFBELT program instruction

SELECT program instruction

SELECT real-valued function

WINDOW program instruction

WINDOW real-valued function

BXOR operator

Syntax

... value BXOR value ...

Function

Perform the **binary** exclusive-OR operation on two values.

Usage Considerations

The BXOR operation is meaningful only when performed on integer values. Only the integer parts of real values are used. Any fractional parts are ignored.

Details

The BXOR operator can be used to perform a binary exclusive-OR operation on two values on a bit-by-bit basis, resulting in a real value.

> **NOTE:** This operation is meaningful only when performed on integer values.

Specifically, the BXOR operation consists of the following steps:

1. Convert the operands to sign-extended 32-bit integers, truncating any fractional part.
2. Perform a binary exclusive-OR operation (see below).
3. Convert the result back to a real value.

During the binary exclusive-OR operation,

```
c = a BXOR b
```

the bits in the resultant C are determined by comparing the corresponding bits in the operands A and B, as indicated in the following table.

For each bit in:			
a	**b**		**c**
0	0	->	0
0	1	->	1
1	0	->	1

For each bit in:			
a	**b**		**c**
1	1	->	0

That is, a bit in the result is 1 if the corresponding bit in one (and only one) of the operands is 1. To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Examples

Consider the following (binary values are shown only to make the operation more evident):

^B101000 BXOR ^B100001 yields ^B001001 (9)

Note that a very different result is obtained with the logical XOR operation:

^B101000 XOR ^B100001 yields 0 (FALSE)

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

BAND operator

BOR operator

XOR operator

BY keyword

Syntax

SCALE(transformation BY *value*)

SHIFT(transformation BY *value, value, value*)

Function

Complete the syntax of the SCALE and SHIFT functions.

Examples

```
SET new.trans = SCALE(old.trans BY scale.factor)
SET new.trans = SHIFT(old.trans BY x,y,z)
```

Related Keywords

SCALE transformation function

SHIFT transformation function

CALIBRATE program instruction

Syntax

CALIBRATE mode, status

Function

Initialize the robot positioning system with the robot's current position.

Usage Considerations

Normally, the instruction is issued with mode equal to zero.

The instruction has no effect if the DRY.RUN system switch is enabled.

If the robot is to be moved under program control, the CALIBRATE instruction (or command) must be processed every time system power is turned on and the eV+ system is booted.

The robot cannot be moved under program control or with the pendant until a CALIBRATE instruction (or CALIBRATE monitor command) has been processed.

> **NOTE:** Some robots can be moved in joint mode with the control pendant even when they have not been calibrated.

If multiple robots are connected to the system controller, this instruction attempts to calibrate all robots unless they are disabled with the ROBOT switch. All of the enabled robots must be calibrated before any of them can be moved under program control.

The CALIBRATE instruction may operate differently for each type of robot. For robots with non-absolute (e.g., incremental) encoders, this instruction causes the robot to move. In this case, the robot must be far enough from the limits of the working range that it will not move out of range during the calibration process. (See the description of the CALIBRATE monitor command for details of the robot motion.)

If the eV+ system is not configured to control a robot, executing the CALIBRATE instruction causes an error.

Parameters

mode — A real expression that indicates what part of calibration is to be performed:

Value of mode	Interpretation
0 (or omitted)	Perform a normal calibration of all the robots controlled by the system. In detail, the following operations are performed: (a) Load the main calibration program if it is not already in memory. (b) Execute the main calibration program with the load, execute, and delete flags set. That causes the robot-specific calibration routines to be loaded, the robots to be calibrated, and the robot routines to be deleted. (Note that the main calibration program is left in memory.)
1	Load the main calibration program if it is not already in memory, and execute the main calibration program with the load flag set. That causes the calibration program to load the applicable robot-specific calibration routines. Note, however, that the calibration process is not performed.
2	Execute the main calibration program (which must already be in memory) with the execute flag set. That causes the system robot(s) to be calibrated, and all the calibration programs to be left in memory.
3	Execute the main calibration program (which must already be in memory) with the delete flag set. That causes the calibration program to delete the robot-specific calibration routines from memory. Note, however, that the actual calibration process is not performed, and the main calibration program is left in memory.

status — Real-valued variable that receives the exit status returned by the calibration program, or (in mode -1) from eV+ when trying to enter into

the special "calibrate" mode.

Details

When started, eV+ assumes that the robot is not calibrated and restricts your ability to move the robot with the pendant or an application program.

> **NOTE:** The COMP mode light on the pendant does not come on when the robot is not calibrated.

Robots with incremental encoders lose start-up calibration whenever system power is switched off. As a safety measure, these robots also lose start-up calibration whenever an *Encoder quadrature error* occurs for one of the robot joints. Other servo errors that can cause the robot to lose calibration are *Unexpected zero index*, *No zero index*, and *RSC Communications Failure*.

- For the Cobra s600 and s800 robots, this instruction causes a small motion of joint 4 (theta).

If this program instruction attempts to load the main calibration program, the same program, module, and file name, and search algorithm, are employed as for the CALIBRATE monitor command.

If you wish to carry out a CALIBRATE instruction in task 0, one way to do so is from a program run using the /C qualifier on the EXECUTE instruction. With that qualifier specified, a program to calibrate the robot can run in task 0 even when DRY.RUN is disabled. A program running in any task other than 0 can execute the CALIBRATE instruction without special conditions.

Example

The following instruction sequence can be used by any program task to perform start-up calibration on the robot (if task #0 is used, the DRY.RUN switch must be enabled before the program is executed):

```
DETACH()              ;Detach the robot
DISABLE DRY.RUN       ;Ensure DRY.RUN is disabled
CALIBRATE             ;Calibrate the robot
ATTACH()              ;Reattach the robot
```

Related Keywords

CALIBRATE monitor command

NOT.CALIBRATED system parameter

SELECT program instruction

SELECT real-valued function

CALL program instruction

Syntax

CALL program(arg_list**)**

Function

Suspend execution of the current program and continue execution with a new program (that is, a subroutine).

Parameters

program Name of the new program to be executed.

arg_list Optional list of subroutine arguments (separated by commas) to be passed between the current program and the new program. (If no argument list is specified, the parentheses after the program parameter can be omitted.)

Arguments can be used to pass data to the called program, to receive results back, or a combination of both. (How arguments are passed is described below.)

Each argument can be any one of the data types supported by eV+ (that is, belt, precision point, real-value, string, and transformation), and can be specified as a constant, a variable, or an expression.[1] The type of each argument must match the type of its counterpart in the argument list of the called program. An argument specified as a variable can be a simple variable, an array element, or an array with one or more of its indexes left blank. (See below for more information.)

> **NOTE:** If a value is being passed back to the calling program, the parameter must be specified as a variable.

Any argument can be omitted, with the result that the corresponding argument in the called program will be undefined. If an argument is omitted within the argument list, the separating comma must still be included. If an argument is omitted at the end of the list, the comma preceding the argument can also be omitted. (See the description of .PROGRAM for more information on the effect of omitting an argument.)

Details

The CALL instruction causes execution of the current program to be suspended temporarily. Execution continues at the first step of the indicated new program, which is then considered a subroutine.

Execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Execution continues with the instruction immediately following the CALL instruction.

Subroutine arguments can be passed by value or by reference. When an argument is passed by value, a copy of the argument value is passed to the subroutine. Any changes to the corresponding subroutine argument in the subroutine will **not** be passed back to the calling routine. Any argument that is specified as an expression (or compound transformation) will be passed by value.

When an argument is a (scalar or array) variable, it is passed by reference. That means a **pointer** to the variable is passed to the subroutine, which then works with exactly the same variable as the calling routine. If the called routine changes the value of the variable, the value is also changed for the calling routine. This can be especially significant, for example, if the same variable is passed as two arguments of a subroutine call. Then, any change to either of the corresponding subroutine arguments in the subroutine automatically changes the other corresponding subroutine argument.

Note that an argument that is passed by reference (because it is a variable) can generally be forced to passage by value. The way that is done depends on the type of the variable, as follows:

- For a real variable, passage by value can be forced by enclosing the variable in parentheses:

```
CALL prog_a((count))
```

- For a string variable, an empty string ("") can be added to the variable:

```
CALL prog_b($str.var+"")
```

- For a transformation variable (for example, start), an equivalent transformation value can be specified by a compound transformation consisting of the variable and the NULL transformation:

```
CALL prog_c(start:NULL)
```

As stated above, the items in the arg_list must match their corresponding items in the called program. In addition to straightforward matches of scalar to scalar, and arrays of equal numbers of dimensions, there are several situations in which higher dimension arrays can be passed in place of lower dimension arrays. For example, all the following cases are valid:

- Array element passed to a scalar:

```
CALL example(a[1])        --->       .PROGRAM example(b)
```

```
CALL example(a[1,2])      --->      .PROGRAM example(b)
CALL example(a[1,2,3])    --->      .PROGRAM example(b)
```

- One dimension of an array passed to a one-dimensional array:

```
CALL example(a[])         --->      .PROGRAM example(b[])
CALL example(a[1,])       --->      .PROGRAM example(b[])
CALL example(a[1,2,])     --->      .PROGRAM example(b[])
```

- Two dimensions of an array passed to a two-dimensional array:

```
CALL example(a[,])        --->      .PROGRAM example(b[,])
CALL example(a[1,,])      --->      .PROGRAM example(b[,])
```

- Three dimensions of an array passed to a three-dimensional array:

```
CALL example(a[,,])       --->      .PROGRAM example(b[,,])
```

Examples

```
CALL pallet(count)
```

Branches to the program named pallet, passing to it a pointer to the variable count. When a RETURN instruction is executed, control returns to the program containing the CALL instruction and count will contain the current value of the corresponding subroutine argument.

```
CALL cycle(1, , n+3)
```

Branches to the program named cycle. The value 1 is passed to the first parameter of cycle, its second parameter is undefined, and its third parameter receives the value of the expression n+3. (If cycle has more than three parameters, the remaining parameters are all undefined.)

Because none of the arguments in the CALL are variables, no data will be returned by the program cycle.

Related Keywords

CALLP program instruction

CALLS program instruction

.PROGRAM program instruction

RETURN program instruction

CALLP program instruction

Syntax

CALLP var (arg_list)

Function

Call a program given a pointer to the program in memory.

Usage Considerations

Using SYMBOL.PTR and CALLP is an alternative to using CALLS to invoke a eV+ subroutine given its name as string data. For some applications, the SYMBOL.PTR-CALLP combination is much faster than CALLS.

Parameters

var	A real variable (not an expression) that contains a pointer to a program symbol.
arg_list	A list of arguments to be passed between the current program and the new program.

Details

In situations where the same program is called multiple times, the CALLP instruction can be significantly more efficient than the CALLS instruction. This is especially true in systems that have many programs loaded. (In situations where a program is called only once, the CALLS instruction is faster.)

When a CALLS instruction is used, the following steps are performed *each time* the CALLS instruction is encountered:

1. The user string is evaluated.
2. The eV+ program symbol with that name is found in the eV+ symbol table.
3. The proper eV+ program is called.

As an alternative, the SYMBOL.PTR function can be used to perform the first two steps. Typically, that is done one time, during the initialization portion of the application software. Then, in place of the CALLS instruction, a CALLP instruction can be used to perform the third step above. (The CALLP instruction is just slightly slower than a CALL instruction.)

In situations where the same program is called multiple times, avoiding the first two steps of CALLS can be significant, especially in systems that have many programs loaded.

The CALLP instruction calls the program pointed to by the real variable **var**. This variable should have been obtained by using the SYMBOL.PTR function. If the value of **var** is zero, no program is called, and no error is reported. If **var** does not contain a valid pointer, program execution stops with error -406 (*Undefined program or variable name*).

Example

Instructions such as the following are executed in the initialization section of the application program:

```
my.pgm.ptr[1] = SYMBOL.PTR("my.program.1")
my.pgm.ptr[2] = SYMBOL.PTR("my.program.2")
```

Then, in the repetitive section of the application program, the following is executed:

```
CALLP my.pgm.ptr[index](parm1, parm2)
```

Related Keywords

CALL program instruction

CALLS program instruction

$SYMBOL string function

SYMBOL.PTR real-valued function

CALLS program instruction

Syntax

CALLS string(arg_list**)**

Function

Suspend execution of the current program and continue execution with a new program (that is, a subroutine) specified with a string value.

Usage Considerations

CALLS takes much longer to execute than the normal CALL instruction. Thus, CALLS should be used only when necessary.

Parameters

string	String value, variable, or expression defining the (1- to 15-character) name of the new program to be executed. (The letters in the name can be lowercase or uppercase.)
arg_list	Optional list of arguments to be passed between the current program and the new program. The parentheses can be omitted if no argument list is specified. (See the CALL instruction for further information on subroutine arguments.)

NOTE: Since the argument list is not specified as part of the string parameter, all the subroutines called by a specific CALLS instruction must have equivalent argument lists.

Details

The CALLS instruction behaves exactly as the CALL instruction does. The only difference between the two instructions is the way the subroutine name is specified. CALL requires that the name be explicitly entered in the instruction step. CALLS permits the name to be specified by a string variable or expression, which can have its value defined when the program is executed. That allows the program to call different subroutines depending on the circumstances.

As with the CALL instruction, execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Execution continues with the instruction immediately following the CALLS instruction.

For some applications, the CALLP instruction is much more efficient than CALLS. See the CALLP instruction for details.

Examples

The program segment below demonstrates how the CALLS instruction can be used to branch to a subroutine whose name is determined when the program is executed.

First the program reads a set of four digital input lines (1001 to 1004) to determine which of sixteen different part types it is dealing with. The part type is considered to be a hexadecimal number, which is converted to the corresponding ASCII character. Once the character is defined, the appropriate subroutine (that is, part.0, part.1, ..., part.f) is called to process the part. (The part-type value is also used to select the portion of the two-dimensional array argument that is passed to the subroutine.)

```
type = BITS(1001,4)                          ;Get part type from digital input
$type = $ENCODE(/H0, type)                   ;Convert to ASCII
character
CALLS "part."+$type(arguments[type,], status)
```

This example can be made more robust by using the STATUS real-valued function to make sure the proposed subroutine exists before it is called. Using this function avoids possible errors from undefined program names.

Related Keywords

CALL program instruction

CALLP program instruction

.PROGRAM program instruction

RETURN program instruction

CAS real-valued function

Syntax

CAS (variable, test_value, new_value)

Function

This function compares a real variable to a test value, and conditionally sets a new value as one indivisible operation.

Usage Considerations

The eV+ system does not enforce any protection scheme for global variables that are shared by multiple program tasks. It is the programmer's responsibility to keep track of the usage of such global variables. The CAS real-valued function (or the similar TAS function) can be used to implement logical interlocks on access to shared variables.

This function can also be used to work around a restriction on the simultaneous access of global arrays by multiple program tasks — program execution can fail if two or more tasks attempt to increase the size of an array at the same time. For a detailed description of this, see the "Global Array Access Restriction" section of the information about Arrays in the *eV+ Language User's Guide*.

Parameters

variable	Name of the real-valued variable to be tested and assigned the new value given.
test_value	Real value, variable, or expression that defines the comparison value.
new_value	Real value, variable, or expression that defines the new value to be assigned to the specified variable.

Details

If the variable is equal to the test value, the new value is stored in the variable. Otherwise the variable is not modified. The original value of the variable is returned as the function value.

The compare and set-new-value operations occur with interrupts locked so that the operation is indivisible. This function provides a way for setting semaphores between tasks, similar to the TAS real-valued function. See the description of that function for more information — use of the CAS function is similar.

If the variable is undefined when the function is executed, it is treated as having the value zero.

Related Keywords

TAS real-valued function

CASE program instruction

Syntax

CASE value *OF*

Function

Initiate processing of a CASE structure by defining the value of interest.

Usage Considerations

This instruction must be part of a complete CASE structure.

Parameter

value	Real value, variable, or expression that defines the value to be matched in the CASE structure to determine which instructions are executed.

Details

This is perhaps the most powerful structure available with eV+. It provides a means for executing one group of instructions from among any number of groups. The complete syntax is as follows (the blank lines are not required):

```
CASE value OF
  VALUE value_1,...:
    group_of_steps
  VALUE value_2,...:
    group_of_steps
      .
      .
      .
  ANY
    group_of_steps
END
```

The three vertical dots indicate that any number of VALUE steps can be used to set off additional groups of instructions.

The ANY step and the group of steps following it are optional. There can be only one ANY step in a CASE structure, and it must mark the last group in the structure (as shown above).

Note that the ANY and END steps must be on lines by themselves as shown. (However, as with all instructions, those lines can have comments.)

The CASE structure is processed as follows:

The expression following the CASE keyword is evaluated.

All the VALUE steps are scanned until the first one is found that has the same value.

The group of instructions following that VALUE step is executed.

Execution continues at the first instruction after the END step.

If no VALUE step is found that contains the same value as that in the CASE instruction, and there is an ANY step in the structure, then the group of instructions following the ANY step will be executed.

If no VALUE match is found in the structure, and there is no ANY step, none of the instructions in the entire CASE structure are executed.

Examples

The following example shows the basic function of a CASE statement:

```
CASE number OF
  VALUE 1:
    TYPE "one"
  VALUE 2:
    TYPE "two"
  ANY
    TYPE "Not one or two"
END
```

The following sample program asks you to enter a test value. If the value is negative, the program exits after displaying a message. Otherwise, a CASE structure is used to classify the input value as a member of one of three groups. The groups are (1) even integers from zero to ten, (2) odd integers from one to nine, and (3) all other positive numbers.

```
PROMPT "Enter a value from 1 to 10: ", x
CASE x OF
  VALUE 0, 2, 4, 6, 8, 10:
    TYPE "The number", x, " is EVEN"
  VALUE 1, 3, 5, 7, 9:
    TYPE "The number", x, " is ODD"
  ANY
    TYPE x, " is not an integer from 0 to 10"
END
```

The following example shows a special use of the CASE structure to test Boolean conditions:

```
PROMPT "Enter a number", x
CASE TRUE OF
  VALUE (x > 0):
    TYPE "The number was greater than 0."
  VALUE (x == 0):
    TYPE "The number was equal to 0."
  VALUE (x < 0):
    TYPE "The number was less than 0."
```

```
END
```

Related Keywords

ANY program instruction

END program instruction

VALUE program instruction

$CHR string function

Syntax

$CHR (value)

Function

Return a one-character string corresponding to a given ASCII value.

Parameter

value Real-valued expression defining the value to be translated into a character. The value must be in the range of 0 to 255 (decimal).

If the value is in the range 0 to 127 (decimal), the corresponding ASCII character will be returned.

Example

```
$CHR(65)     ;Returns the character A, since its ASCII value is
65.
```

Related Keywords

ASC real-valued function

$DBLB string function

$FLTB string function

$INTB string function

$LNGB string function

CLEAR.EVENT program instruction

Syntax

CLEAR.EVENT task, flag

Function

Clear an event associated with the specified task.

Parameters

task	Optional real value, variable, or expression (interpreted as an integer) that specifies the task for which the event is to be cleared. The valid range is 0 to 6 or 0 to 27, inclusive. If this parameter is omitted, the number of the current task is used.
flag	Not used, defaults to 1.

> **NOTE:** The basic system allows 7 tasks (0 - 6). The eV+ Extensions option allows 28 tasks (0 - 27).

Details

This instruction clears the event associated with the specified task.

The default event cleared is the input/output completion event for which the instruction WAIT.EVENT 1 waits. This event is also cleared by the execution (not the completion) of an input/output instruction.

Related Keywords

GET.EVENT real-valued function

SET.EVENT program instruction

WAIT.EVENT program instruction

CLOSE and CLOSEI program instruction

Syntax

CLOSE

CLOSEI

Function

Close the robot gripper.

Usage Considerations

CLOSE causes the hand to close during the next robot motion.

CLOSEI causes a BREAK in the current continuous-path motion and causes the hand to close immediately after the current motion completes.

The CLOSE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The CLOSEI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions send a signal to the control valves for the pneumatic hand to close. If the CLOSE instruction is used, the signal is not sent until the next robot motion begins.

The CLOSEI instruction differs from CLOSE in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion, or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Examples

During the next robot motion, cause the pneumatic control valves to assume the closed state:

```
CLOSE
```

Cause the pneumatic control valves to assume the closed state as soon as the current motion stops:

```
CLOSEI
```

Related Keywords

HAND.TIME system parameter

OPEN program instruction

OPENI program instruction

RELAX program instruction

RELAXI program instruction

SELECT program instruction

SELECT real-valued function

CLOSE and CLOSEI program instruction

Syntax

CLOSE

CLOSEI

Function

Close the robot gripper.

Usage Considerations

CLOSE causes the hand to close during the next robot motion.

CLOSEI causes a BREAK in the current continuous-path motion and causes the hand to close immediately after the current motion completes.

The CLOSE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The CLOSEI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions send a signal to the control valves for the pneumatic hand to close. If the CLOSE instruction is used, the signal is not sent until the next robot motion begins.

The CLOSEI instruction differs from CLOSE in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion, or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Examples

During the next robot motion, cause the pneumatic control valves to assume the closed state:

```
CLOSE
```

Cause the pneumatic control valves to assume the closed state as soon as the current motion stops:

```
CLOSEI
```

Related Keywords

HAND.TIME system parameter

OPEN program instruction

OPENI program instruction

RELAX program instruction

RELAXI program instruction

SELECT program instruction

SELECT real-valued function

COARSE program instruction

Syntax

COARSE tolerance *ALWAYS*

Function

Enable a low-precision feature of the robot hardware servo.

Usage Considerations

Only the next robot motion will be affected unless the ALWAYS parameter is specified.

If the tolerance parameter is specified, its value becomes the default for any subsequent COARSE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified).

FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The COARSE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the COARSE instruction causes an error.

Parameters

tolerance	Optional real value, variable, or expression that specifies the percentage of the standard coarse tolerances that are to be used for each joint of the robot attached by the current execution task. See the Details section for default values.
ALWAYS	Optional qualifier that establishes COARSE as the default condition. That is, COARSE will remain in effect until disabled by a FINE instruction. If ALWAYS is not specified, the COARSE instruction will apply only to the next robot motion.

Details

This instruction enables a low-precision feature in the robot motion servo so that larger errors in the final positions of the robot joints are permitted at the ends of motions. This allows faster motion execution when high accuracy is not required.

Lower precision is sometimes required in belt tracking applications when the constant motion of the robot prevents the servos from settling to high precision.

If the tolerance parameter is specified, the new setting takes effect at the start of the next motion. Also, the value becomes the default for any subsequent COARSE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified). Changing the COARSE tolerance does not affect the FINE tolerance.

If the tolerance parameter is omitted, the most recent setting (for the attached robot) is used. The default setting is restored to 100 percent when program execution begins, or a new execution cycle starts (assuming that the robot is attached to the program).

Examples

Enable the low-precision feature only for the next motion:

```
COARSE
```

Enable the low-precision feature for the next motion, with the tolerance settings changed to 150% of the standard tolerance for each joint (that is, a looser tolerance):

```
COARSE 150
```

Enable the low-precision feature until it is explicitly disabled:

```
COARSE ALWAYS
```

Related Keywords

CONFIG real-valued function

DELAY.IN.TOL program instruction

FINE program instruction

NONULL program instruction

NULL program instruction

SELECT program instruction

SELECT real-valued function

COM operator

Syntax

... COM value ...

Function

Perform the binary complement operation on a value.

Usage Considerations

The word "com" cannot be used as a program name or variable name.

The COM operation is meaningful only when performed on an integer value. Only the integer parts of real values are used. Any fractional parts are ignored.

Parameter

value	Real-valued expression defining the value to be complemented.

Details

The COM operator performs the binary complement operation on a bit-by-bit basis, resulting in a real value.

Specifically, the COM operation consists of the following steps:

1. Convert the operand to a sign-extended 32-bit integer, truncating any fractional part.
2. Perform a binary complement operation.
3. Convert the result back to a real value.

To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Examples

For example:

```
COM 40    yields   -41
```

Note that a very different result is obtained with the **logical** complement operation (NOT):

```
NOT 40    yields    0.0    (FALSE)
```

In this case, 40 is interpreted as logically TRUE since it is nonzero.

CONFIG real-valued function

Syntax

CONFIG (select**)**

Function

Return a value that provides information about the robot's geometric configuration, or the status of the motion servo-control features.

Usage Considerations

The CONFIG function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the CONFIG function causes an error.

Parameter

select Optional real value, variable, or expression (interpreted as an integer) that has a value from 0 to 13 and selects the category of the configuration information to be returned. (See below for details.)

Details

This function returns a value that is interpreted as a series of bit flags. The interpretation of the value returned by this function depends on the select parameter.

When the select parameter is omitted, or has the value 0, 1, or 2, the CONFIG function returns a value that can be interpreted as bit flags indicating a geometric configuration of the robot. That is, each bit in the value represents one characteristic of a robot configuration.

The parameter values in this group determine which robot configuration is returned by the function:

Select	Configuration information returned
0	The robot's current (instantaneous) configuration. (The default value is 0.)
1	The configuration the robot will achieve at the completion of the current motion, or the current configuration if no motion is in progress (and the robot is attached).

Select	Configuration information returned
	NOTE: The result returned is not meaningful if the robot is not attached.
2	The configuration the robot achieves at the completion of the next motion (assuming that it is a joint-interpolated [not straight-line] motion).

The interpretations of the bit flags returned by these selections are as follows:

Bit #	Bit Mas-k	Indication if bit SET
1	1	Robot has righty configuration
2	2	Robot has below configuration
3	4	Robot has flipped configuration

When the select parameter is 3, 4, or 5, the CONFIG function returns a value that can be interpreted as bit flags indicating the settings of several robot motion servo-control features. That is, each bit in the value represents the state of one motion servo-control feature.

The different parameter values in this group select which motion(s) will be affected by the features settings reported by the function, as follows:

Select	Configuration information returned
3	The permanent settings of the robot motion servo-control features. That is, the settings defined by instructions that specify the ALWAYS qualifier.
4	The temporary settings for the motion currently executing, or the last motion completed if no motion is in progress.
5	The temporary settings that will apply to the next motion performed.

The interpretations of the bit flags returned by selections 3, 4, and 5 are as follows:

Bit#	Bit Mask	Indicati-on if bit CLEAR	Bit SET
1	1	(None)	(None)
2	2	FINE asserted	COARSE asserted
3	4	NULL asserted	NONULL asserted
4	8	MULTIPL-E asserted	SINGLE asserted
5	^H10	CPON asserted	CPOFF asserted
6	^H20	OVERLA-P asserted	NOOVER-LAP asserted

When the select parameter is 6, 7, or 8, the CONFIG function returns a value that represents the setting of the FINE tolerance.

Select	FINE tolerance returned
6	The permanent setting, as a percentage of the standard tolerance.
7	The setting used for the previous or current motion, as a percentage of the standard tolerance.
8	The setting to be used for the next motion, as a percentage of the standard tolerance.

When the select parameter is 9, 10, or 11, the CONFIG function returns a value that represents the setting of the COARSE tolerance.

Select	COARSE tolerance returned
9	The permanent setting, as a percentage of the standard tolerance.
10	The setting used for the previous or current motion, as a percentage of the standard tolerance.
11	The setting to be used for the next motion, as a percentage of the standard tolerance.

When the select parameter is 12, the available joint configuration options for the selected robot are returned as shown below.

Bit #	Bit Mask	Indication if bit set
1	1	Robot can have lefty or righty configuration.
2	2	Robot can have above or below configuration.
3	4	Robot can have flipped or noflip configuration.
18	^H20000	Robot supports the OVERLAP and NOOVERLAP instructions.
22	^H200000	Robot's last rotary joint can be limited to ±180 degrees

Bit #	Bit Mask	Indication if bit set
		(SINGLE or MULTIPLE).

When the select parameter is 13, the type of robot motion is returned. The bit values returned are shown below.

Bit #	Bit Mask	Description
1	1	This bit is set if the motion is joint interpolated; it is cleared for straight-line motion.
2	2	This bit is set if the robot is performing a SPIN motion.

Related Keywords

ABOVE program instruction

BELOW program instruction

COARSE program instruction

CPOFF program instruction

CPON program instruction

FINE program instruction

FLIP program instruction

LEFTY program instruction

MULTIPLE program instruction

NOFLIP program instruction

NONULL program instruction

NOOVERLAP program instruction

NULL program instruction

OVERLAP program instruction

RIGHTY program instruction

SELECT program instruction

SELECT real-valued function

SINGLE program instruction

STATE real-valued function

COS real-valued function

Syntax

COS (angle)

Function

Return the trigonometric cosine of a given angle.

Usage Considerations

The angle parameter must be measured in degrees.

The parameter is interpreted as modulo 360 degrees, but excessively large values may cause a loss of accuracy in the returned value.

Parameter

angle	Real-valued expression that defines the angular value (in degrees) to be considered.

Details

Returns the trigonometric cosine of the argument, which is assumed to be in degrees. The resulting value is always in the range of -1.0 to +1.0, inclusive.

Examples

```
COS(0.5)              ;Returns 0.999962
COS(-5.462)           ;Returns 0.9954596
COS(60)               ;Returns 0.4999999
COS(1.3125E+2)        ;Returns -0.6593457
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

CP system switch

Syntax

... CP

Function

Control the continuous-path feature.

Details

The CP switch can be used to turn off continuous-path motion processing. For more information on continuous path motion, see the section Continuous Path Trajectories in the *eV+ Language User's Guide*.

This switch is *enabled* when the eV+ system is initialized.

Example

```
DISABLE CP      ;Turn off continuous-path motion processing.
```

Related Keywords

BREAK program instruction

CPOFF program instruction

CPON program instruction

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

CPOFF program instruction

Syntax

CPOFF *ALWAYS*

Function

Instruct the eV+ system to stop the robot at the completion of the next motion instruction (or all subsequent motion instructions) and null position errors.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The CPOFF instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies only to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the CPOFF instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes CPOFF as the default condition. That is, when ALWAYS is included in a CPOFF instruction, CPOFF will remain in effect continuously until disabled by a CPON instruction. If ALWAYS is not specified, the CPOFF instruction applies only to the next robot motion.

Details

When CPOFF is in effect, the robot will be brought to a stop at the completion of the next robot motion, and any final position errors will be nulled (if required).

Unlike the BREAK instruction, which is executed *after* a motion to cause continuous-path processing to terminate, CPON and CPOFF are executed *before* a motion instruction to affect the continuous-path processing of the next motion instruction. Also, while BREAK applies to only one motion instruction, CPOFF can apply to all the motion instructions that follow.

> **NOTE:** The BREAK instruction causes continuous-path processing to terminate by blocking eV+ program execution until the motion ends. CPOFF causes the trajectory generator to terminate continuous path without affecting the forward processing of the eV+ program.

If the CP system switch is disabled, continuous-path processing never occurs regardless of any CPON or CPOFF instructions.

Related Keywords

BREAK program instruction

CP system switch

CPON program instruction

SELECT program instruction

SELECT real-valued function

CPON program instruction

Syntax

CPON *ALWAYS*

Function

Instruct the eV+ system to execute the next motion instruction (or all subsequent motion instructions) as part of a continuous path.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

This is the default state of the eV+ system. CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The CPON instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies only to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the CPON instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes CPON as the default condition. That is, if ALWAYS is specified, CPON will remain in effect continuously until disabled by a CPOFF instruction. If ALWAYS is not specified, the CPON instruction applies only to the next robot motion.

Details

When CPON is in effect, it is possible to execute a series of motion instructions that are blended into a single continuous path. That is, each motion will be performed in succession without stopping the robot at specified locations.

Unlike the BREAK instruction, which is executed *after* a motion to cause continuous-path processing to terminate, CPON and CPOFF are executed *before* a motion instruction to affect the continuous-path processing of the next motion instruction.

> **NOTE:** The BREAK instruction causes continuous-path processing to terminate by blocking eV+ program execution until the motion ends. CPOFF causes the trajectory generator to terminate continuous path without affecting the forward processing of the eV+ program.

While asserting CPON permits continuous-path processing to occur, any of the following conditions will break a continuous path and override CPON:

- No subsequent motion instruction is executed before completion of the next motion instruction.
- CP system switch is disabled.

  (If the CP system switch is disabled, continuous-path processing never occurs, regardless of any CPON or CPOFF instructions.)
- The next motion instruction is followed by an instruction that explicitly or implicitly causes motion termination (for example, BREAK, OPENI).

Related Keywords

BREAK program instruction

CP system switch

CPOFF program instruction

SELECT program instruction

SELECT real-valued function

CYCLE.END program instruction

Syntax

CYCLE.END task_num, stop_flag

Function

Terminate the executing program in the specified task the next time it executes a STOP program instruction (or its equivalent).

Suspend processing of an executable program until a program running in the specified task completes execution.

Usage Considerations

The CYCLE.END instruction has no effect if the specified program task is not active.

The CYCLE.END instruction suspends execution of the program containing the instruction until the specified program task completes execution. If a program is aborted while its execution is suspended by a CYCLE.END instruction, the program task specified by the CYCLE.END instruction will still be terminated (if the stop_flag is TRUE).

Parameters

task_num	Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be monitored or terminated. If the task number is not specified, the CYCLE.END *instruction* always accesses task #0.
stop_flag	Optional real value, variable, or expression interpreted as a logical (TRUE or FALSE) value. If the parameter is omitted or has the value 0, the specified task is *not* stopped-but the CYCLE.END has all its other effects (see below). If the parameter has a nonzero value, the selected task stops at the end of its current cycle.

Details

If the stop_flag parameter has a TRUE value, the specified program task will terminate the next time it executes a STOP program instruction (or its equivalent), regardless of how many program cycles are left to be executed.

> **NOTE:** CYCLE.END will not terminate a program with continuous internal loops. Such a program must be terminated with the ABORT command or instruction.

Regardless of the stop_flag parameter, this instruction will wait until the program actually is terminated. If the program being terminated loops internally so that the current execution cycle never ends, the CYCLE.END instruction will wait forever.

To proceed from a CYCLE.END that is waiting for a program to terminate, abort the program that is waiting for a CYCLE.END by typing an ABORT command for the program task that executed the CYCLE.END instruction.

Example

The following program segment shows how a program task can be initiated from another program task (the ABORT and CYCLE.END program instructions are used to make sure the specified program task is not already active):

```
ABORT 3                 ;Abort any program already active
CYCLE.END 3             ;Wait for execution to abort
EXECUTE 3 new.program   ;Start up the new program
```

Related Keywords

ABORT monitor command

ABORT program instruction

CYCLE.END monitor command

EXECUTE monitor command

EXECUTE program instruction

KILL monitor command

KILL program instruction

STATUS monitor command

STATUS real-valued function

STOP program instruction

DBLB real-valued function

Syntax

DBLB ($string, first_char**)**

Function

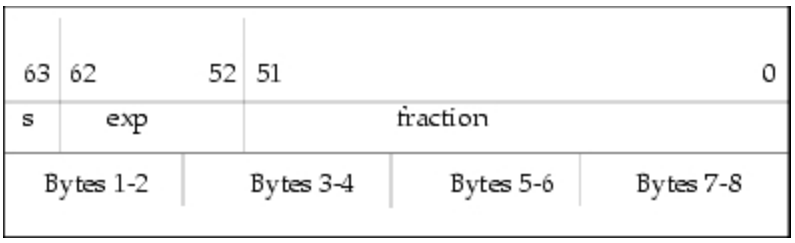
Return the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.

Parameters

$string	String expression that contains the eight bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the eight bytes in the string. If first_char is omitted or has a value of 0 or 1, the first eight bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second through ninth bytes are extracted. If first_char specifies eight bytes that are beyond the end of the input string, an error is generated .

Details

Eight sequential bytes of the given string are interpreted as being a double-precision (64-bit) floating-point number in the IEEE standard format. This 64-bit field is interpreted as follows:



where

s is the sign bit, s = 0 for positive, s = 1 for negative.

exp is the binary exponent, biased by -1023.

fraction is a binary fraction with an implied 1 to the left of the binary point.

For 0 < exp < 2047, the value of a floating point number is:

$$-1^{s} * (1.fraction) * 2^{exp\ -1023}$$

Double-precision real values have the following special values:

exp	fraction	Description
0	Zero	Zero value
0	Nonzero	Denormalized number
2047	Zero	Signed infinity
2047	Nonzero	Not-a-number

The range for normalized numbers is approximately 2.2×10^{-308} to 1.8×10^{307}

The main use of this function is to convert a binary floating-point number from an input data record to a value that can be used internally by eV+.

Example

```
DBLB($CHR(^H3F)+$CHR(^HF0)+$CHR(0)+$CHR(0)
+$CHR(0)+$CHR(0)+$CHR(0)+$CHR(0))            ;Returns 1.0
```

Related Keywords

ASC real-valued function

$DBLB string function

DOUBLE (type keyword for AUTO, GLOBAL, and LOCAL)

FLTB real-valued function

$FLTB string function

INTB real-valued function

TRANSB transformation function

VAL real-valued function

$DBLB string function

Syntax

$DBLB (value)

Function

Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.

Parameter

value	Real-valued expression, the value of which is converted to its IEEE floating-point binary representation.

Details

A real value is converted to its binary representation using the IEEE double-precision standard floating-point format. This 64-bit value is packed as eight successive 8-bit characters in a string. See the DBLB real-valued function for a more detailed description of IEEE floating-point format.

The main use of this function is to convert a double-precision real value to its binary representation in an output record of a data file.

Example

```
$DBLB(1.215)
```

Returns a character string equivalent to:

```
$CHR(^H3F)+$CHR(^H3F)+$CHR(^H70)+
$CHR(^HA3)+$CHR(^HD7)+$CHR(^H0A)+$CHR(^H3D)+$CHR(^H71)
```

Related Keywords

$CHR string function

DOUBLE (type keyword for AUTO, GLOBAL, and LOCAL)

$FLTB string function

FLTB real-valued function

$INTB string function

$TRANSB string function

DCB real-valued function

Syntax

DCB (value)

Function

Convert BCD digits into an equivalent integer value.

Usage Considerations

No more than four BCD digits can be converted.

The DCB function is most often used with the BITS real-valued function to decode input from the digital input signal lines.

Parameter

value	Real value interpreted as a binary bit pattern representing up to four BCD digits.

> **NOTE:** An error is reported if any digit is not a valid BCD digit. That is, if a digit is greater than 9.

Example

If external input signals 1001-1008 (8 bits of input) receive two BCD digits from an external device, then the instruction

```
input = DCB(BITS(1001, 8))
```

sets the real variable input equal to the integer equivalent of the BCD input on the specified signals.

Related Keyword

BCD real-valued function

DECEL.100 system switch

Syntax

... DECEL.100[robot_num]

Function

Enable or disable the use of 100 percent as the maximum deceleration for the ACCEL program instruction.

Parameter

robot_num	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

When DECEL.100 is enabled for the selected robot, the maximum deceleration percentage defined by the SPEC utility program is ignored, and a maximum deceleration of 100% is used instead. This maximum is used to limit the value specified by the ACCEL program instruction. By default, DECEL.100 is disabled for all robots.

Example

```
DECEL.100[2]      ;Cause ACCEL to use 100% for maximum
                  ;deceleration for robot #2
```

Related Keywords

ACCEL program instruction

ACCEL real-valued function

SPEED monitor command

SPEED program instruction

$DECODE string function

Syntax

$DECODE ($string_var, string_exp, mode**)**

Function

Extract part of a string as delimited by given break characters.

Usage Considerations

$DECODE modifies the input *$string_var* variable as well as returning a string value.

The test for break characters is always performed without regard for the case of the characters in the input string.

The break characters are treated as individual characters, independent of the other characters in the string that defines them.

Parameters

$string_ var String variable that contains the string to be scanned. After the function is processed, this variable will contain the portion of the original string that was *not* returned as the function value.

NOTE: This parameter is modified by the function and cannot be specified as a string constant or expression.

If the program causes the same variable to receive the function value, the variable will end up containing the value returned by the function.

string_exp String constant, variable, or expression that defines the individual break characters, which are to be considered as separating the substrings of interest in the input string value. (The order of the characters in this string has no effect on the function operation.)

mode Optional real value, variable, or expression that controls the operation performed by the function. Mode values are -3, -2, 0, and 1.

If the mode is negative or zero, or is omitted, characters up to the first break character are removed from the input string and returned as the output of the function.

If the mode is greater than zero, characters up to the first *non*break character are removed from the input string and returned as the

output of the function. That is, this case returns all the leading break characters in the input string.

Details

This function is used to scan an input string and return the initial substring, as delimited by any of a group of break characters. After the substring is returned by the function, it is *deleted* from the input string.

The string returned (and deleted) can either contain no break characters (mode 0), or nothing but break characters (mode 1). That is, $DECODE can return (and delete) all the characters up to the first break character-usually some desired substring; or the function can return (and delete) all the leading break characters-which are usually discarded.

By alternating the modes, groups of desired characters can be picked from the input string (see the first example below).

The modes -2 and -3 copy all nonbreak characters up to the first break characters plus the first break character. Mode -2 is equivalent to the following instructions:

```
$s = $DECODE($i,$break,0)    ;Extract up to 1st break character
$s = $s+$MID($i,1,1)         ;Add on 1st break character
$i = $MID($i,2,127)          ;Remove the break character
```

The following instruction can perform these operations:

```
$s = $DECODE($i,$break,-2)   ;Extract through 1st break character
```

Mode -3 is equivalent to mode -2 if a break character is present. However, if no break character is contained in the input string, mode -3 returns an empty string and leaves the input string unchanged.

Examples

The instructions below pick off consecutive numbers from the string $input, assuming that the numbers are separated by some combination of spaces and commas.

The first instruction within the DO structure sets the variable $temp to the substring from $input that contains the first number (and removes that substring from $input). The VAL function is used to convert the numeric string into its corresponding real value, which is assigned to the next element of the real array value. The $DECODE function is used a second time to extract the characters that separate the numbers (the characters found are ignored).

```
i = 0                                          ;Set array index
DO
    $temp = $DECODE($string_var," ,",0)        ;Pick off a number
string
    value[i] = VAL($temp)                      ;Convert to real value
    $temp = $DECODE($string_var," ,",1)        ;Discard spaces and
```

```
commas
    i = i+1                                 ;Advance the array
index
UNTIL $string_var == ""                     ;Stop when input is
empty
```

In a case where *$string_var* contains a sequence of numeric values (as strings) separated by spaces, commas, or any combination of spaces and commas, such as

```
$string_var = "1234. 93465.2, .4358,3458103"
```

executing the above instructions results in the first four elements of the value array having the following values:

```
value[0] = 1234.0
value[1] = 93465.2
value[2] = 0.4358
value[3] = 3458103.0
```

The string variable input ($string_var) also contains an empty string ("").

As shown above, use of the $DECODE function normally involves two string variables: the input variable and the output variable. If you are interested only in the characters up to the first break character, and want to discard all the characters that follow, the same variable can be used for both input and output. In the following instruction, for example, the same variable is used on both sides of the equal sign because the programmer wants to discard all the white space (that is, space and tab) characters at the end of the input string.

> **NOTE:** The break characters are specified by a string expression consisting of a space character and a tab character.

```
$line = $DECODE($line," "+$CHR(9),0)    ;Discard trailing blanks
```

Related Keywords

$TRUNCATE string function

$MID string function

DECOMPOSE program instruction

Syntax

DECOMPOSE array_name[index**] = location**

Function

Extract the (real) values of individual components of a location value.

Parameters

array_name	Name of the real-valued array that has its elements defined.
index	Optional integer value(s) that identifies the first array element to be defined. Zero will be assumed for any omitted index. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.
location	Location value that is decomposed into its component values. This can be a transformation value or a precision-point value, and can be defined by a variable or a location-valued function.

Details

This instruction assigns values to consecutive elements of the named array, corresponding to the components of the specified location.

If the location is represented as a transformation value, six elements are defined, corresponding to X, Y, Z, yaw, pitch, and roll.

If the location is represented as a precision-point value, then from one to twelve elements are defined (depending on the number of robot joints), that correspond to the individual joint positions.

Examples

The following code assigns the components of transformation part to elements 0 to 5 of array x:

```
DECOMPOSE x[] = part
```

The following code assigns the components of precision point #pick to array element angles [4] and those that follow it:

```
DECOMPOSE angles[4] = #pick
```

Related Keywords

#PPOINT precision-point function

TRANS transformation function

$DEFAULT string function

Syntax

$DEFAULT (mode**)**

Function

Return a string containing the current or initial system default device, unit, and directory path for disk file access.

Usage Considerations

Parentheses must be included even when mode is omitted.

Parameter

mode — Optional real value, variable, or expression (interpreted as an integer) that specifies the default path to be returned, as follows:

- If the parameter is omitted, or has the value zero, the current system default path is returned.
- If the parameter has the value one, the default path returned is the one that was in effect immediately after the eV+ system was booted from disk.

Details

The system default device, unit, and directory path can be set by the CD or DEFAULT monitor command. The $DEFAULT function returns the current or initial default values as a string. The string contains the portions of the following information that have been set:

```
device>disk_unit:directory_path
```

where

device — is one of the following:

DISK	a local disk
SYSTEM	a disk device, drive, and subdirectory path currently set with the DEFAULT command

For more details on valid devices, see the ATTACH program instruction on page 65.

disk_unit	is the disk unit specified to the DEFAULT monitor command. The colon (:) is omitted if no unit was specified.
directory_path	is any input to the DEFAULT command that followed the device and unit. The directory path is omitted if no additional input was specified.

Example

The following commands set the default drive specification to DISK>D:\TEST\, and then display it on the terminal for confirmation:

```
DEFAULT = DISK>D:\TEST\ LISTS $DEFAULT()
```

Related Keywords

CD monitor command

DEFAULT monitor command

DEFBELT program instruction

Syntax

DEFBELT %belt_var = nom_trans, belt_num, vel_avg, scale_fact

Function

Define a belt variable for use with a conveyor tracking robot.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The DEFBELT program instruction supports up to six belt encoders, depending on the hardware configuration.

The BELT switch must be enabled for this instruction to be executed.

The DEFBELT program instruction cannot be executed while the robot is moving relative to the specified belt variable.

When a belt variable is initialized using this instruction, its window parameters are set to allow any location in the working volume of the robot. That is, no belt window is set. (See the WINDOW program instruction.)

When a belt variable is initialized with the DEFBELT program instruction, error checking is initiated for the associated belt encoder. This error checking can be turned off by disabling the BELT system switch or by using the ZERO command to reinitialize the eV+ system.

Parameters

%belt_var Name of the belt variable to be defined. (All appearances of belt variables must be prefixed with the percent character [%].)

nom_trans Transformation value that defines the position and orientation of the conveyor belt. This can be provided by a transformation variable, a transformation-valued function, or a compound transformation.

The X axis of the nominal transformation defines the direction of travel of the belt. Normally, the belt moves along the direction of +X. The X-Y plane defined by this transformation is parallel to the surface of the belt. The (X, Y, Z) position defined by the nominal transformation defines the approximate center of the belt with respect to the robot.

belt_num The number of the encoder used for reading the instantaneous

	location of the belt. Belts numbered from 1 to 6 can be specified. This can be specified with a constant, a variable, or an expression.
vel_avg	(This parameter is currently ignored, but some value must be provided.)
scale_fact	The calibration constant that relates motion of the conveyor belt with counts of the encoder mounted on the conveyor. This value (which can be supplied as a constant, a real variable, or an expression) is interpreted as having the units in millimeters of belt travel per encoder count.

Details

A conveyor belt is modeled by a belt variable. In addition to the parameters for the DEFBELT program instruction, a belt variable contains the following information:

- Window parameters, which define the working range of the robot along the belt. (Set with the WINDOW instruction.)
- An encoder offset, which is used to adjust the origin of the belt frame of reference. (Set with the SETBELT instruction.)

Belt variables have the following characteristics:

- Belt variable names must always be preceded by the percent character (%), for example, %main.belt. Otherwise, the normal rules for variable names apply.
- Belt variable arrays are allowed, for example, %b[x].
- Belt variables can be passed as subroutine parameters just like other variables.
- Belt variables can be defined only with the DEFBELT instruction-there is no assignment instruction for them. Thus, the following are *not* valid instructions:

```
%new_belt = %old_belt
SET %new_belt = %old_belt
```

- Belt variables cannot be stored on a mass-storage device. (Variables used to define the parameters in a DEFBELT instruction can be stored, however.)

Example

The following instruction defines the belt variable %belt.var. The value of b.num must be the number of the encoder to be associated with this belt variable. The variable b.num is also used as an index for arrays of data describing the position and orientation of the belt, its velocity smoothing, and the encoder scale factor.

```
DEFBELT %belt.var = belt.nom[b.num], b.num, v.avg[b.num], belt.sf
```

```
[b.num]
```

Related Keywords

BELT system switch

BELT system switch

BELT.MODE system parameter

BSTATUS real-valued function

SETBELT program instruction

WINDOW program instruction

WINDOW real-valued function

DEFINED real-valued function

Syntax

DEFINED (var_name)

Function

Determine whether a variable has been defined.

Parameter

var_name	The name of a location, string, or real variable. Both scalar variables and array variables are permitted. A location variable can be a transformation, a precision point, or a belt variable.

Details

The value of the specified variable is tested. If the value is defined, the function returns the value TRUE. Otherwise, the value FALSE is returned.

For array variables, if a specific array element is specified, the single array element is tested. If no array element is specified, this function returns a TRUE value if any element of the array is defined.

> **NOTE:** For nonreal arguments (i.e., strings, locations, transformations) that are passed in the argument list of a CALL statement, you can test to see if the variable is defined or not. However, you cannot assign a value to undefined nonreal arguments within the CALLed program. If you attempt to assign a value to an undefined nonreal argument, you receive an undefined value error message.

Therefore, when using DEFINED to test for user input, be sure to assign a default value to the variable before testing it. See the example below.

Examples

```
.PROGRAM test($s)
        AUTO $tmp
        $tmp = "default"
        IF DEFINED($s) THEN
            $tmp = $s
        END
        TYPE /C3 "The string is: ", $tmp
```

When the example above is executed with a value:

```
ex test("ABCD")
```

the routine returns:

```
The string is: ABCD
```

When the example above is executed without a value:

```
ex test()
```

the routine returns:

```
The string is: default
```

The instruction:

```
DEFINED(base_part)
```

returns a value of TRUE if the variable base_part is defined.

The instruction:

```
DEFINED(corner[])
```

returns a value of TRUE if any element of the array corner has been defined.

Related Keywords

STATUS real-valued function

TESTP monitor command

DELAY program instruction

Syntax

DELAY time

Function

Cause robot motion to stop for the specified time.

Usage Considerations

The robot stops during the delay, but the wait and nulling normally associated with a motion BREAK do not occur.

Program execution continues during the delay, up to the next motion instruction in the program. (eV+ system timers can be used to control the timing of program execution. The DELAY instruction should not be used for that purpose.)

The DELAY instruction is interpreted as a straight-line move-to-here motion instruction. (See below for the consequences of that interpretation.)

The DELAY instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the DELAY instruction causes an error.

If the AMOVE instruction has been executed to prepare for motion of an extra axis, execution of the DELAY instruction cancels the effect of the AMOVE instruction.

Parameter

time	Real value, variable, or expression that specifies the length of time, in seconds, that the robot is to pause. A time value less than 0.016 (16 milliseconds) results in a 16-millisecond delay.

Details

The DELAY instruction is processed as a robot motion. As a result, the following consequences occur when a DELAY is executed:

1. Any pending hand actuation takes place during the execution of the DELAY instruction.
2. Any temporary trajectory switches that have been specified are cleared after the

conclusion of the delay.

3. Any pending configuration change is canceled.

> **NOTE:** When DRY.RUN mode is in effect, DELAY instructions do not cause any delay.

Examples

```
DELAY 2.5
```

Causes all robot motion to stop for 2.5 seconds and any pending hand operation to occur. Clears any temporary trajectory switches that may be set, and cancels any pending requests for configuration change.

```
DELAY pause.1
```

Stops all robot motion for pause.1 seconds.

Related Keywords

DURATION program instruction

SELECT program instruction

SELECT real-valued function

DELAY.IN.TOL system switch

Syntax

... DELAY.IN.TOL [robot_num]

Function

Controls the timing of COARSE or FINE nulling after eV+ completes a motion segment.

Parameter

robot_num	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

The DELAY.IN.TOL system switch is **disabled** by default for all robot device modules, except the Delta robot device module.

If the switch is disabled, COARSE or FINE nulling completes whenever eV+ has completed a motion segment and the robot is tracking the **trajectory** to within the coarse or fine tolerance. The actual robot location might not be within the tolerance of the endpoint.

If the switch is enabled, COARSE or FINE nulling completes whenever eV+ has completed a motion segment and the actual robot location is within the specified coarse or fine tolerance of the **endpoint**of that motion segment.

Usage Considerations

For many applications, enabling this switch produces the best nulling behavior. However, the switch should be disabled for backward compatibility with previous eV+ systems.

Related Keywords

COARSE program instruction

FINE program instruction

NULL program instruction

NONULL program instruction

DELAY.POWER.OFF system switch

Syntax

... DELAY.POWER.OFF

Function

Enable/disable the ESTOP timer delay feature for servo errors.

Usage Considerations

This switch is only operational for systems equipped with an AWC-II board as the main CPU. For program compatibility, the DELAY.POWER.OFF system switch is recognized by eV+ systems for both AWC-II-based controllers and SmartController systems, but the switch has no effect on the latter.

Details

When DELAY.POWER.OFF is disabled (default), servo errors will cause the robot power to be disabled immediately, without an ESTOP timer delay.

When DELAY.POWER.OFF is enabled, servo errors will use the ESTOP timer delay function just as if the ESTOP button had been pressed.

DEPART and DEPARTS program instruction

Syntax

DEPART distance

DEPARTS distance

Function

Start a robot motion away from the current location.

Usage Considerations

DEPART causes a joint-interpolated motion.

DEPARTS causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Parameter

distance Real-valued expression that specifies the distance (in millimeters) along the robot tool Z axis between the current robot location and the desired destination.

A positive distance moves the tool back (toward negative tool Z) from the current location; a negative distance moves the tool forward (toward positive tool Z).

Details

These instructions initiate a robot motion to a new location, which is offset from the current location by the distance given, measured along the current tool Z axis.

Examples

```
DEPART 80
```

Moves the robot tool 80 millimeters back from its current location using a joint-interpolated motion.

```
DEPARTS 2*offset
```

Withdraws the robot tool (2 * offset) millimeters along a straight-line path from its current location.

Related Keywords

APPRO program instruction

APPROS program instruction

MOVE program instruction

MOVES program instruction

SELECT program instruction

SELECT real-valued function

DEPART and DEPARTS program instruction

Syntax

DEPART distance

DEPARTS distance

Function

Start a robot motion away from the current location.

Usage Considerations

DEPART causes a joint-interpolated motion.

DEPARTS causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Parameter

distance Real-valued expression that specifies the distance (in millimeters) along the robot tool Z axis between the current robot location and the desired destination.

A positive distance moves the tool back (toward negative tool Z) from the current location; a negative distance moves the tool forward (toward positive tool Z).

Details

These instructions initiate a robot motion to a new location, which is offset from the current location by the distance given, measured along the current tool Z axis.

Examples

```
DEPART 80
```

Moves the robot tool 80 millimeters back from its current location using a joint-interpolated motion.

```
DEPARTS 2*offset
```

Withdraws the robot tool (2 * offset) millimeters along a straight-line path from its current location.

Related Keywords

APPRO program instruction

APPROS program instruction

MOVE program instruction

MOVES program instruction

SELECT program instruction

SELECT real-valued function

DEST transformation function

Syntax

DEST

Function

Return a transformation value representing the planned destination location for the current robot motion.

Usage Considerations

The DEST function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the DEST function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word "dest" cannot be used as a program name or variable name.

Details

DEST returns the location to which a robot was moving when its motion was interrupted. This applies for all motion instructions, including:

1. Motions to named locations, such as

   ```
   MOVE start
   MOVES #part[10]
   ```

   Note that even though the second instruction references a precision-point location variable, the DEST function returns a transformation value during that motion.

2. Motions to locations defined relative to named locations or defined relative to the current robot location

   ```
   APPROS drop, 50.00
   DEPART 30.00
   MOVE SHIFT(HERE BY 50,0,10)
   ```

3. Motions to special locations such as

   ```
   READY
   ```

The location value returned by the DEST function may not be the same as the location at which the robot stops if the motion of the robot is interrupted for some reason. For example, if the RUN/HOLD button on the pendant is pressed, the robot stops immediately, but DEST still returns the location to which the robot was moving.

If a motion is not begun because eV+ realizes the destination location cannot be reached (for example, it is too far from the robot), then DEST is **not** set to the goal location.

Example

The DEST function is useful, for example, for continuing a motion that has been interrupted by a reaction initiated by a REACTI instruction. The subroutine automatically invoked can contain steps such as the following to process the interruption and resume the original motion.

```
SET save = HERE          ;Record where the robot is
SET old.dest = DEST      ;Record where the robot was going
old.speed = SPEED(3)     ;Record the current motion speed
DEPART 50.0              ;Back away a safe distance
       .
       .
       .
APPRO save, 50.0         ;Return to the original motion path
MOVES save               ;...back to where we left
SPEED old.speed          ;Restore the original motion speed
MOVES old.dest           ;Continue toward original destination
```

Related Keywords

HERE transformation function

#PDEST precision-point function

SELECT program instruction

SELECT real-valued function

DETACH program instruction

Syntax

DETACH (logical_unit**)**

Function

Release a specified device from the control of the application program.

Usage Considerations

Detaching the robot causes a BREAK in continuous-path motion.

DETACH automatically forces an FCLOSE if a disk file or graphics window is open on the specified device.

The robot is automatically attached to task 0 when the EXECUTE monitor command or program instruction is processed to initiate that task and the DRY.RUN system switch is disabled. All the other logical units are automatically detached when program execution begins. (Other events that cause automatic detachment are listed below.)

Parameter

logical_unit	Optional real value, variable, or expression (interpreted as an integer) that identifies the device to be detached. (See the ATTACH instruction for a description of logical unit numbers.) The parentheses can be omitted if the logical unit number is omitted (causing the robot to be detached).

Details

This instruction releases the specified device from control by the application program. (No error is generated if the device was not attached.)

Control of the specified device can be returned to the program with the ATTACH instruction.

When logical_unit is 0 (or is omitted), the program releases control of the robot. While the robot is detached, robot power can be turned off and on, the pendant can be used to move the robot, and a different robot can be selected (if more than one robot is connected to the system controller). A delay of one system cycle (16 ms) occurs when a robot is detached.

This is useful for applications that require you to define where the robot should be located for certain operations. For such tasks a teaching program can DETACH the robot and then output directions to you on the system terminal or the pendant. You can then use the pendant to move the robot to the desired locations. The system terminal or the pendant can be used for accepting input from you (the latter can be read by using the PENDANT function).

When a disk logical unit is detached, any device that was specified by the corresponding ATTACH instruction is forgotten. Thus, a subsequent ATTACH instruction must specify the device again if the default device is not desired.

The following events automatically DETACH all the logical units (except the robot) from the affected program task:

- Processing of the EXECUTE command and instruction
- Processing of the KILL command and instruction
- Processing of the ZERO command
- Normal completion of program execution

Note, however, that if a program terminates execution "abnormally", all of its devices remain attached, except that the terminal and the manual control pendant are detached. (Abnormal termination of program execution refers to any cause other than HALT or STOP instructions.) If the task is subsequently resumed, the program automatically reattaches the terminal and pendant if they were attached before the termination.

> **NOTE:** It is possible that another program task attached the terminal or pendant in the meantime. This results in an error message when the stopped task is restarted.

```
DETACH          ;Release program control of the robot.
DETACH (1)      ;Discontinue program control of the
                ;manual control pendant.
```

Related Keyword

ATTACH program instruction

DEVICE program instruction

Syntax

DEVICE (type, unit, error, p1, p2, ...**)** out[i], in[j], out_trans, in_trans

Function

Send a command or data to an external device and, optionally, return data back to the program. (The actual operation performed depends on the device referenced.)

Usage Considerations

The syntax contains optional parameters that apply only to specific device types and commands.

Parameters

type — Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced. The following types are currently available:

0 = Belt encoder
1 = (Not used)
2 = Force Processor Board (for Adept use only)
3 = Robot device (i.e., servo, for Adept use only)
4 = Vision
5 = 1394 bus (for Adept use only)

unit — Real value, variable, or expression (interpreted as an integer) that indicates the device unit number. The value must be in the range 0 to (max -1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.

error — Optional real variable that receives a standard system error number if this instruction failed. If this parameter is omitted, any device error stops program execution. If the error parameter is specified, the program must explicitly check it to detect errors.

p1, p2, ... — Optional real values, variables, or expressions, the values of which are sent to the device as part of the command. The number of parameters specified, and their meanings, depend upon the particular device type being accessed.

out[]	Optional real array that contains data values that are sent to the device as part of a command. The actual data sent depends upon the device type and command being sent to the device.
i	Optional real value, variable, or expression (interpreted as an integer) that indicates the first array element to be considered in the array out []. Element 0 is accessed first if no index is specified.
in[]	Optional real array that receives any data values returned from the device as the result of the command. The actual data returned depends upon the device type and the command.
j	Optional real value, variable, or expression (interpreted as an integer) that indicates the first array element to be filled in the array in[]. Element 0 is accessed first if no index is specified.
out_ trans	Optional transformation variable, function, or compound transformation that defines a transformation value to be sent to the device as part of the command. The actual data sent depends on the device type and the command.
in_trans	Optional transformation variable that receives a data value returned from the device as the result of a command. The actual data returned depends upon the device type and the command.

Details

DEVICE is a general-purpose instruction for accessing external devices. For more information and examples, see the section External Encoder Device in the *eV+ Language User's Guide*.)

Related Keywords

DEVICE real-valued function

DEVICES program instruction

SETDEVICE program instruction

DEVICE real-valued function

Syntax

DEVICE (type, unit,error, p1, p2,...**)**

Function

Return a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.

Usage Considerations

The syntax contains optional parameters that may be useful only for specific device types and information requests.

Parameters

type	Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced. The following types are currently available: 0 = Belt encoder 1 = (Not used) 2 = Force Processor Board (for Adept use only) 3 = Robot device (i.e., servo, for Adept use only) 4 = Vision 5 = 1394 bus (for Adept use only)
unit	Real value that indicates the device unit number. The value must be in the range 0 to (max -1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real variable that receives a standard system error number, which indicates if this function succeeded or failed. If this parameter is omitted, any device error stops program execution. If error is specified, the program must check it to detect errors.
p1, p2, ...	Optional real values that are sent to the device as part of the request. The number of values specified and the meanings of the values depend upon the particular device type.

Details

DEVICE is a general-purpose function for returning data and status information from external devices. For details and examples see the supplementary documentation for specific devices.

For information on use of the DEVICE function to access external encoders, see the section External Encoder Device in the *eV+ Language User's Guide*.

For systems equipped with AdeptSight, the DEVICE instruction is used to configure vision system memory allocation and frame buffer configuration.

Related Keywords

DEVICE program instruction

DEVICES program instruction

SETDEVICE program instruction

DEVICES program instruction

Syntax

DEVICES (type, unit, error, p1, p2, ...**)** $out, $in

Function

Send a command or data to an external device and optionally return data. The actual operation performed depends on the device referenced.

Usage Considerations

The syntax contains optional parameters that may be useful only for specific device types and commands.

Parameters

type Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced. The following types are currently available:

0 = Belt encoder
1 = (Not used)
2 = Force Processor Board (for Adept use only)
3 = Robot device (i.e., servo, for Adept use only)
4 = Vision
5 = 1394 bus (for Adept use only)

unit Real value that indicates the device unit number. The value must be in the range 0 to (max -1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.

error Optional real variable that receives a standard system error number if this instruction failed. If this parameter is omitted, any device error stops program execution. If the error parameter is specified, the program must check it to detect errors. The value is negative if there was an error. Otherwise, the positive value indicates the number of data bytes that were returned in the **$in** parameter.

p1, p2, ... Optional real values that are sent to the device as part of a command. The number of values specified and the meanings of the values depend upon the particular device type.

$out	Optional string expression, variable, or array that defines a string value to be sent to the device as part of the command. The actual data that should be sent depends upon the device type and the command. When the $out parameter is specified as an array, the total length of the string value must be less than or equal to 520 bytes. If an array is specified, and no index is specified, element 0 is accessed first.
$in	Optional string variable or array that receives any data values returned from a device as the result of the command. The actual data returned depends upon the device type and the command. The error variable receives the number of input bytes returned. When the $in parameter is specified as an array, up to 512 bytes may be returned, packed in up to four successive string array elements. If an array is specified, and no index is specified, element 0 is accessed first.

Details

DEVICES is a general-purpose instruction for accessing external devices. It is similar to the DEVICE program instruction except that data items are sent and received as strings rather than real values.

> **NOTE:** Similar to the CALL and CALLS instruction pair, this instruction is a string-based version of the DEVICE instruction. Thus, the name DEVICES can be thought of as "device s", rather than the plural of "device".

For details and examples see the supplementary documentation for specific devices.

Related Keywords

DEVICE program instruction

DEVICE real-valued function

SETDEVICE program instruction

DISABLE program instruction

Syntax

DISABLE switch, ... , switch

Function

Turn off one or more system control switches.

Usage Considerations

If a specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the switch array are enabled.

Parameter

switch Name of a system switch to be turned off.

The name can be abbreviated to the minimum length that uniquely identifies the switch. That is, for example, the MESSAGES switch can be referred to with ME since there is no other switch with a name beginning with the letters ME.

Details

System switches control various aspects of the operation of the eV+ system, including some optional subsystems such as vision. The Switch entry in the index for this document directs you to the detailed descriptions of these switches.

Other system switches are available when options are installed. Refer to the option documentation for details.

When a switch is disabled, or turned off, the feature it controls is no longer functional or available for use. Turning a switch on with the ENABLE monitor command or program instruction makes the associated feature functional or available for use.

> **NOTE:** The system switches are shared by all the program tasks. Thus, care should be exercised when multiple tasks are disabling and enabling switches-otherwise, the switches may not be set correctly for one or more of the tasks. Disabling the DRY.RUN switch does not have an effect until the next EXECUTE command or instruction is processed for task #0, an ATTACH instruction is executed for the robot, or a CALIBRATE command or instruction is processed.

The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a switch at any time. The SWITCH program instruction can be used, like the DISABLE instruction, to disable a switch.

Example

```
DISABLE MESSAGES        ;Turns off the MESSAGES switch.
```

Related Keywords

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

DISTANCE real-valued function

Syntax

DISTANCE (location_1, location_2)

Function

Determine the distance between the points defined by two location values.

Parameter

location_1 Transformation value that defines the first point of interest. This can be a function, a variable, or a compound transformation.

location_2 Transformation value that defines the second point of interest. This can be a function, a variable, or a compound transformation.

Details

Returns the distance in millimeters between the points defined by the two specified locations. The order in which the locations are specified does not matter. Also, the orientations defined by the locations have no effect on the value returned.

Example

The statement

```
x = DISTANCE(HERE, part)
```

sets the value of the real variable x to be the distance between where the robot tool point is currently located and the point defined by the transformation part.

Related Keyword

IDENTICAL real-valued function

DN.RESTART program instruction

Syntax

DN.RESTART

Function

Restarts DeviceNet communication if the CanBus goes offline.

Details

The eV+ DeviceNet interface goes offline if you disconnect it from the actual network or if too many errors occur during operation. DN.RESTART forces eV+ to reinitialize the DeviceNet interface as if you rebooted your system.

Related Keywords

DEVICENET monitor command

DN.RESTART monitor command

DO program instruction

Syntax

DO

Function

Introduce a DO program structure.

Usage Considerations

The DO program structure must be concluded with an UNTIL instruction.

Details

The DO structure provides a way to control the execution of a group of instructions based on a control expression. The syntax for the DO structure is as follows:

```
DO
    group_of_steps
UNTIL logical_expression
```

Processing of the DO structure can be described as follows:

1. The group of instruction steps is executed.
2. The logical expression is evaluated. If the result is FALSE, return to item 1. Otherwise, proceed to item 3.
3. Program execution continues at the first instruction after the UNTIL step.

When this structure is used, it is assumed that some action occurs within the group of enclosed instructions that changes the result of the logical expression from TRUE to FALSE when the structure should be exited. Alternately, logical_expression can be replaced with an expression that evaluates the state of a digital I/O signal (see example).

Note that the group of instructions within the DO structure is *always* executed at least one time. (The WHILE structure differs in that respect.)

There do not need to be any instructions between the DO and UNTIL instructions. When there are no such instructions, the UNTIL criterion is continuously evaluated until it is satisfied, at which time program execution continues with the instructions following the UNTIL instruction.

Example

The following example uses a DO structure to control a task that involves moving parts from one place to another. The sequence assumes that the digital signal line buffer.full changes to

the on state when the parts buffer becomes full. (The robot should then perform a different sequence of motions.)

```
.
.
DO
    CALL get.part()
    CALL put.part()
UNTIL SIG(buffer.full)
.
.
```

Related Keywords

DO monitor command

EXIT program instruction

NEXT program instruction

UNTIL program instruction

WHILE program instruction

DOS program instruction

Syntax

DOS string, error

Function

Execute a program instruction defined by a string expression.

Usage Considerations

Before the instruction is executed, the string must be translated from ASCII into the internal representation used by eV+. Thus, the instruction *executes much more slowly* than a normal program instruction.

The string cannot define a declaration statement or most of the control structure statements.

The DOS instruction is ignored if the string defines a comment line or a blank line.

Parameters

string	String constant, variable, or expression that defines the program instruction to be executed. The instruction may contain a label field (which is ignored) and may be followed by a standard comment field. Leading and trailing spaces and tabs are ignored.
error	Optional real variable that receives any parsing or execution error generated by the instruction. The value is set to 1 if the instruction succeeds. If the instruction fails, a standard eV+ error number is returned. If this parameter is omitted and an error occurs, execution of the program stops and the appropriate error message is displayed.

Details

The DOS (DO String) instruction provides a means for modifying a program on the fly. That is, the embedded program instruction, which is defined by a string expression, is executed as though it had been entered in the program as a normal instruction.

The instruction executes in the context of the current program. Thus, any subroutine argument, automatic variable, or local variable can be accessed.

If a variable referenced in the instruction is not found in the current program context, the variable is assumed to be global. Any new variables that are created by the instruction (for

example, in an assignment statement) are created as globals. Normal variable type checking is performed, and errors are generated if there are type conflicts.

The single-line control statements GOTO, IF ... GOTO, CALL, and CALLS are allowed and execute normally. The multiline control structures (for example, CASE ... END, IF ... ELSE ... END) cannot be executed by the DOS instruction.

Examples

```
DOS "var = 123"
```

Causes the variable var to be assigned the value 123. If var is undefined, a new *global* variable named var is created. Any errors cause the program to stop executing.

```
DOS $ins, status
```

Causes the instruction contained in the string variable $ins to be executed. If an error occurs, a eV+ error code is placed in the real variable status and execution continues.

Related Keywords

DO monitor command

MCS program instruction

DRIVE program instruction

Syntax

DRIVE joint, change, speed

Function

Move an individual joint of the robot.

Usage Considerations

The DRIVE instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the DRIVE instruction causes an error.

If the AMOVE instruction has been executed to prepare for motion of an extra axis, execution of the DRIVE instruction cancels the effect of the AMOVE instruction.

Parameters

joint	Number of the robot joint to be moved. This can be specified by a constant, a variable, or an expression.
change	The change desired in the joint position. This can be specified by a constant, a variable, or an expression. The value can be positive or negative. The value is interpreted in the units used to measure the joint position. That is, a change for a rotary joint must be the number of degrees the joint is to move; a change for a linear joint must specify the number of millimeters to move.
speed	The temporary program speed to be used for the motion, considered as a percentage of the current monitor speed setting. Again, this can be specified by a constant, a variable, or an expression.

Details

Operates the single specified robot joint, changing its position by **change** amount (in degrees or millimeters). The joint number, **joint**, can be 1, 2, ..., n, where n is the number of joints the robot has.

The speed of the motion is governed by a combination of the speed given in this instruction and the monitor SPEED setting. That is, the regular program speed setting is not used. (See

the SPEED monitor command and the SPEED program instruction for explanations of motion speeds.)

The duration setting established by the DURATION instruction also affects the execution time of the motion.

Example

Change the angle of joint 2 by driving the joint 62.4 degrees in the negative direction at a speed of 75% of the monitor speed:

```
DRIVE 2,-62.4,75
```

Related Keywords

SELECT program instruction

SELECT real-valued function

DRY.RUN system switch

Syntax

... DRY.RUN

Function

Control whether or not eV+ communicates with the robot.

Usage Considerations

The DRY.RUN switch can be enabled or disabled by an application program, but the new setting of the switch does not take effect until the next time any of the following events occur:

1. An EXECUTE command or instruction is processed for task #0
2. The robot is attached with an ATTACH instruction
3. A CALIBRATE command or instruction is processed

Before an application program changes the setting of the DRY.RUN switch, the program must have the robot detached. Otherwise, an error results when the attempt is made to change the switch setting.

Details

This system switch can be used to stop eV+ from sending motion commands to the robot and expecting position information back from the robot. Thus, when the system is in DRY.RUN mode, application programs can be executed to test for such things as proper logical flow and correct external communication without having to worry about the robot running into something. (Also see the TRACE system switch.)

The pendant can still be used to control the robot while the system is in DRY.RUN mode.

The DRY.RUN switch is sampled whenever a robot is attached. (Note that task #0 attempts to attach the robot when program execution begins or is resumed.) The DRY.RUN setting for a task can be changed during execution by DETACHing the robot, changing DRY.RUN, and then ATTACHing the robot.

> **NOTE:** Do not allow multiple tasks to change DRY.RUN simultaneously, since the DRY.RUN state can then be different from that expected. Your programs should use a software interlock in this case.

The DRY.RUN switch is initially **disabled**.

WARNING: Digital and analog I/O is not affected by DRY.RUN, so external devices driven by analog or digital output instructions still operate.

Related Keywords

DISABLE monitor command

ENABLE monitor command

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

DURATION program instruction

Syntax

DURATION time *ALWAYS*

Function

Set the minimum execution time for subsequent robot motions.

Usage Considerations

Unless the ALWAYS parameter is specified, only the next robot motion is affected.

DURATION 0 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The DURATION instruction affects the DRIVE instruction but not the DELAY instruction.

The setting of the monitor SPEED command affects the results of the DURATION setting.

The DURATION instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the DURATION instruction causes an error.

Parameters

time	Real-valued expression that specifies the minimum length of time (in seconds) that subsequent robot motions take to perform (see below). If the value is zero, robot motions are performed without consideration of their time duration and use only the applicable values for SPEED and ACCEL.
ALWAYS	Optional keyword that determines how long the new duration will have an effect. If ALWAYS is included, the specified duration time applies to all subsequent robot motions (until the duration setting is changed by another DURATION instruction). The specified duration applies only to the next robot motion if ALWAYS is not included.

Details

This instruction sets the minimum execution time for subsequent robot motions. For any motion, the time specified by the DURATION instruction has no effect if the duration setting is less than the time computed by the eV+ robot-motion trajectory generator (considering the current motion speed and acceleration settings). However, if the duration is longer than the time computed by the trajectory generator, the motion is slowed so that its elapsed time corresponds approximately to the specified duration.

> **NOTE:** Actual motion times may differ slightly from the duration setting due to quantization effects and due to acceleration and deceleration profiling.

The duration instruction does not specify the duration of an entire motion but instead specifies the minimum time of the constant-velocity segment plus one-half the acceleration and deceleration segments. In this way, continuous-path motions (in which individual motions are blended together) get the correct duration, but a single motion takes *longer* than the specified duration. In other words, the time of motion is primarily defined either by the value of DURATION or SPEED, using whichever value gives the longer time.

This instruction is very useful. Consider, for example, a situation where the value of a periodic, external signal is employed to continuously correct the path of the robot while the robot is moving. The DURATION instruction can be used to match the motion execution time to the sensor sampling rate and processing time. This ensures that the robot is kept in motion while new information is being processed. A sample program of this type is shown later.

Example

The following example reads an external sensor and moves to the computed robot location. This sequence is repeated 20 times at intervals of 96 milliseconds (6/TPS seconds). This assumes the default period (tick) of 16 milliseconds for the eV+ trajectory generator. Note that the motion speed is set to a very large value to make sure the motion is paced by the duration setting.

```
DURATION 6/TPS ALWAYS          ;Each motion to be 6 ticks long
SPEED 200 ALWAYS               ;Motion time determined primarily
                               ;by DURATION, not SPEED
    FOR i = 1 TO 20            ;Repeat 20 times...
        CALL read.signal(loc)  ;Get new step from sensor
        MOVE loc               ;Move to the location
    END
```

Related Keywords

ACCEL program instruction

DELAY program instruction

DURATION real-valued function

SELECT program instruction

SELECT real-valued function

SPEED monitor command

SPEED program instruction

DURATION real-valued function

Syntax

DURATION (select)

Function

Return the current setting of one of the motion DURATION specifications.

Usage Considerations

The DURATION function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, the DURATION function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select	Real-valued expression whose value determines which duration value should be returned (see below).

Details

This function returns the user-specified minimum robot motion duration (in seconds) corresponding to the **select** parameter value. (See the description of the DURATION program instruction for an explanation of the specification of motion duration times.)

Different select values determine when the duration time returned applies, as listed below. (All other values for the **select** parameter are considered invalid.)

select	DURATION value returned
2	Permanent minimum robot motion duration (set by a DURATION ... ALWAYS program instruction)
3	Temporary motion duration for the current or last motion
4	Temporary motion duration to be used for the next motion

Related Keywords

CONFIG real-valued function

DURATION program instruction

SELECT program instruction

SELECT real-valued function

DX, DY, DZ real-valued function

Syntax

DX (location)

DY (location)

DZ (location)

Function

Return a displacement component of a given transformation value.

Parameter

location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

These three functions return the respective displacement components of the specified transformation value.

> **NOTE:** The DECOMPOSE instruction can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, that instruction must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Example

Consider a transformation start with the following components:

```
125, 250, -50, 135, 50, 75
```

The following function references will then yield the indicated values:

```
DX(start)          ;Returns 125.00
DY(start)          ;Returns 250.00
DZ(start)          ;Returns -50.00
```

Related Keywords

DECOMPOSE program instruction

RX transformation function

RY transformation function

RZ transformation function

DX, DY, DZ real-valued function

Syntax

DX (location)

DY (location)

DZ (location)

Function

Return a displacement component of a given transformation value.

Parameter

location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

These three functions return the respective displacement components of the specified transformation value.

> **NOTE:** The DECOMPOSE instruction can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, that instruction must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Example

Consider a transformation start with the following components:

```
125, 250, -50, 135, 50, 75
```

The following function references will then yield the indicated values:

```
DX(start)          ;Returns 125.00
DY(start)          ;Returns 250.00
DZ(start)          ;Returns -50.00
```

Related Keywords

DECOMPOSE program instruction

RX transformation function

RY transformation function

RZ transformation function

DX, DY, DZ real-valued function

Syntax

DX (location)

DY (location)

DZ (location)

Function

Return a displacement component of a given transformation value.

Parameter

location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

These three functions return the respective displacement components of the specified transformation value.

> **NOTE:** The DECOMPOSE instruction can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, that instruction must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Example

Consider a transformation start with the following components:

```
125, 250, -50, 135, 50, 75
```

The following function references will then yield the indicated values:

```
DX(start)          ;Returns 125.00
DY(start)          ;Returns 250.00
DZ(start)          ;Returns -50.00
```

Related Keywords

DECOMPOSE program instruction

RX transformation function

RY transformation function

RZ transformation function

ELSE program instruction

Syntax

ELSE

Function

Separate the alternate group of statements in an IF ... THEN control structure.

Usage Considerations

ELSE can be used only within an IF ... THEN ... ELSE ... END control structure.

Details

Marks the end of a group of statements to be executed if the value of the logical expression in an IF logical_expr THEN control structure is nonzero, and the start of the group of statements to be executed if the value is zero.

Related Keyword

IF ... THEN program instruction

ENABLE program instruction

Syntax

ENABLE switch, ..., switch

Function

Turn on one or more system control switches.

Usage Considerations

The ENABLE monitor command can be used when a program is executing.

If a specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the switch array are enabled.

Parameter

switch Name of a system switch to be turned on.

The name can be abbreviated to the minimum length that uniquely identifies the switch. That is, for example, the MESSAGES switch can be referred to with ME, since there is no other switch with a name beginning with the letters ME.

Details

System switches control various aspects of the operation of the eV+ system, including some optional subsystems such as vision. The Switch entry in the index for this document directs you to the detailed descriptions of these switches.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptSight options are described in the *AdeptSight User's Guide*.

When a switch is enabled, or turned on, the feature it controls is functional and available for use. Turning a switch off with the DISABLE monitor command or program instruction makes the associated feature not functional or available for use.

> **NOTE:** The system switches are shared by all the program tasks. Thus, care should be exercised when multiple tasks are disabling and enabling switches. Otherwise, the switches may not be set correctly for one or more of the tasks.
>
> Disabling the DRY.RUN switch does not have an effect until the next EXECUTE command or instruction is processed for task #0, an ATTACH instruction is executed for the robot, or a CALIBRATE command or instruction is processed.

The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a switch at any time. The SWITCH program instruction can be used, like the ENABLE instruction, to set a switch.

Example

```
ENABLE MESSAGES     ;Turns on the MESSAGES switch.
```

Related Keywords

DISABLE monitor command

DISABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

$ENCODE string function

Syntax

$ENCODE (output_specification, output_specification, ...**)**

Function

Return a string created from output specifications. The string produced is similar to the output of a TYPE instruction.

Parameter

An output specification can consist of any of the following components (in any order) separated by commas:

1. A string expression.
2. A real-valued expression, which is evaluated to determine a value to be displayed.
3. Format-control information, which determines the format of the output message.

Details

This function makes strings normally produced by the TYPE instruction available within a program. That is, $ENCODE does not generate any output but, rather, creates a string value.

The following format specifiers can be used to control the display of numeric values. These settings remain in effect for the remainder of the function parameter list unless another specifier is used.

For all these specifiers, if a value is too large to be displayed in the given field width, the field is filled with asterisk characters (*).

/D	Use the default format, which is equivalent to /G15.8 (see below), except trailing zeros and all but one leading space are omitted.

The following format specifications accept a zero as the width field. This causes the actual field size to vary to fit the value and all leading spaces to be suppressed. This is useful when a value is displayed within a line of text or at the end of a line.

/En.m	Format values in scientific notation (for example, -1.234E+02) in fields n spaces wide with m digits in the fractional parts. If n is not zero, it must be large enough to include space for a minus sign (if the displayed value is negative), one digit to the left of the decimal point, a decimal point (if m is not zero), m digits, and four

	or five characters for the exponent.
/Fn.m	Format values in fixed-point notation (for example, -123.4) in fields n spaces wide, with m digits in the fractional parts.
/Gn.m	Format values in F format with m digits in the fractional parts if that can be done in fields n spaces wide. Otherwise /En.m format is used.
/Hn	Format values as hexadecimal integers in fields n spaces wide.
/In	Format values as decimal integers in fields n spaces wide.
/On	Format values as octal integers in fields n spaces wide.

The following specifiers can be used to insert special characters in the string:

/Cn	Include the characters carriage return (CR) and line feed (LF) n times.

If the string resulting from the $ENCODE function is output to the terminal, this results in n blank lines if the control specifier is at the beginning or end of the function parameter list; otherwise, n -1 blank lines result.

/Un	Include the characters necessary to move the cursor up n lines if the resulting string is output to the terminal. (This works correctly only if the TERMINAL parameter is correctly set for the terminal being used.)
/Xn	Include n spaces.
/B	Include a character that beeps the terminal if the resulting string is output to the terminal.

Example

The program statement:

```
$output = $output+$ENCODE(/F6.2, count)
```

adds a formatted representation of the value of count to the string contained in $output.

The $ENCODE function provides a way of adding format control to the output from PROMPT instructions. This is shown by the following example, in which the value of motor is displayed as part of the prompt message to you.

```
PROMPT $ENCODE(/B,"Start motor #",/I0,motor," (Y/N)? "), $answer
```

This PROMPT instruction beeps the terminal (/B), and displays the following user prompt when the value of motor is 3:

```
Start motor #3 (Y/N)?
```

Related Keyword

TYPE program instruction

END program instruction

Syntax

END

Function

Mark the end of a control structure.

Usage Considerations

Every END instruction must be part of a CASE, FOR, IF, or WHILE control structure.

Details

Every CASE, FOR, IF, and WHILE control structure must have its end marked by an END instruction. The eV+ editor displays an error message when program editing is exited if the correct number of END instructions do not exist in a program (that is, if there are too few or too many).

Related Keywords

CASE program instruction

FOR program instruction

IF ... THEN program instruction

WHILE program instruction

.END keyword

Syntax

.END

Function

Mark the end of a eV+ program.

Usage Considerations

The eV+ editors automatically add this line to the end of every program.

Details

Normally, you will not need to concern yourself with the .END step of programs-it is created automatically by the eV+ editors. The only time you will see this step while working with the eV+ system is when you issue a LISTP monitor command. Then you will see an .END step as the last step of each program.

The .END is important, however, when a program is created on another computer for transfer to a eV+ system. In that case, the programmer must be sure to include a line starting with .END at the end of each program (the remainder of the line is ignored by eV+). Programs missing the .END instruction do not load correctly into the eV+ system.

Related Keyword

.PROGRAM program instruction

ERROR real-valued function

Syntax

ERROR (source, select**)**

Function

Return the error number of a recent error that caused program execution to stop or caused a REACTE reaction.

Usage Considerations

Executing a REACTE statement clears any errors for the current task and prevents the ERROR function from returning errors as expected.

A FIFO buffer is available that receives all asynchronous errors that occur from the time an enable power request is issued (using ENABLE POWER or the MCP) until power is disabled for any reason. The FIFO is accessed using the ERROR( ) real-valued function.

The asynchronous FIFO is not valid while the robot is in the power-down initialization state. User programs should wait until STATE(1) <> 0 before calling ERROR( ) with a **source** parameter > 1000.

Parameters

source Real value, variable, or expression (interpreted as an integer) whose value selects the source of the error code as follows:

-2 Return additional error code for current robot.

ERROR(-2, 0)	returns the standard eV+ error number.
ERROR(-2, 1)	returns the motor mask for the current robot. This bit mask indicates the motor(s) referenced for the error number returned by the instruction ERROR(-2,0). The LSB indicates motor 1, etcetera. If the ERROR(-2, 1)=0, the error is not associated with a specific motor.

-1 Return the number of the most recent error from the program in which the ERROR function is executed.

0 Return the number of the most recent error from the program executing as task #0.

0 < source ≤ 27

Return the number of the most recent error from the program executing as the corresponding task number.

1001 < source < 1021

Asynchronous FIFO element n-1000, where *n* equals

1	Most recent item
2	Next older item, etc.

Returns 0 if no more FIFO elements exist. Valid select parameter values for the FIFO are 0, 1, and 3.

The asynchronous FIFO is not valid while the robot is in power-down initialization state. User programs should wait until STATE(1) returns a nonzero value before using ERROR() with source > 1000.

select Optional real value, variable, or expression (interpreted as an integer) that selects the error information to be returned. (The value 0 is assumed if this parameter is omitted.)

0 Return the error number of the most recent program

execution error (excluding I/O errors-see IOSTAT) for the specified program task.

1 If the most recent error (for the specified program task) had an error code in the range -1100 to -1199, return the variable part of the corresponding error message as a numeric value. If the most recent error had an error code in the range -1000 to -1099, return the variable portion of the corresponding error message as a bit mask indicating the joints or motors to which the error applies. Zero is returned if the error did not have a variable portion in its message. (Also see select = 3 below.)

2 Return the error number of the most recent error from an MCS instruction executed by the specified program task.

3 Return the number of the robot associated with the most recent error for the specified program task. Zero is returned if the error was not associated with a specific robot. (Also see select = 1.)

Details

A eV+ task can access any errors that result in robot power being disabled. These errors include the asynchronous messages that previously were output only to the monitor window.

This function is especially useful in a REACTE subroutine program to determine why the REACTE was triggered.

> **NOTE:** The ERROR function does not report errors reported by the IOSTAT function.

See System Messagesfor a list of all the eV+ error messages and their error numbers.

As noted above, when the select parameter is 1, the value returned by this function should be interpreted as a 6-bit numeric value. The following program illustrates how the value should be interpreted.

Example Program: Return error message corresponding to an error code

```
.PROGRAM error.string(code, vcode, robot, $msg)
; ABSTRACT: Return error message corresponding to error code(s).
;
; INPUTS:        code      Basic error code e.g., from
;                          [e.g., from ERROR(n) or IOSTAT(lun)]
;                vcode     Variable part of the error code
;                          [e.g., from ERROR(n,1)].
```

```
;                robot     Number of the robot associated with the error
;                          [i.e., from ERROR(n,3)].
;
; OUTPUTS:       $msg      Corresponding error message may be null
;
        AUTO i, n
        $msg = ""                                ;Assume no error
        IF code < 0 THEN                         ;If there was an error...
            $msg = $ERROR(code)                  ;Get base message string
        ; Add bit numbers if applicable.
            IF (-1100 < code) AND (code <= -1000) THEN
                n = 1                            ;Initialize bit mask
                FOR i = 1 TO 7                   ;For each of 7 bits
                    IF vcode BAND n THEN         ;If this bit is set,
                        $msg = $msg+$ENCODE(i)   ;add it to message
                    END
                    n = 2*n                      ;Shift the mask 1 bit
                END
           END
        ; Add numeric variable if applicable.
            IF (-1200 < code) AND (code <= -1100) THEN
                $msg = $msg+$ENCODE(vcode) ;Add number
            END
        ; Add robot number if applicable.
            IF robot AND (SELECT(ROBOT,-1) > 1) THEN
                $msg = $msg+" (Robot"+$ENCODE(robot)+")"
            END
        END

        RETURN
.END
```

Related Keywords

$ERROR string function

IOSTAT real-valued function

REACTE program instruction

$ERROR string function

Syntax

$ERROR (error_code)

Function

Return the error message associated with the given error code.

Parameter

error_code	Real-valued expression, with a negative value, that identifies an error condition.

Details

All the error codes returned by the IOSTAT function and by the ERROR real-valued function can be converted into their corresponding eV+ error message strings with this function. (The ERROR real-valued function must be used to determine the *variable* portion of the error message for an error code less than or equal to -1000.)

See System Messages on page 577 for a list of all the eV+ error messages and their error codes.

Example

The following program segment displays an error message if an I/O error occurs:

```
READ (5) $input
IF IOSTAT(5) < 0 THEN
    TYPE "I/O error during read: ", $ERROR(IOSTAT(5))
    HALT
END
```

Related Keywords

ERROR real-valued function

IOSTAT real-valued function

REACTE program instruction

ESTOP program instruction

Syntax

ESTOP

Function

Assert the emergency-stop signal to stop the robot.

Details

This instruction immediately asserts the controller emergency-stop signal and then proceeds with a normal power-down sequence. It is functionally identical to pressing the E-STOP button on the pendant or Front Panel .

Related Keywords

BRAKE program instruction

ESTOP monitor command

PANIC monitor command

PANIC program instruction

STATE real-valued function

EXECUTE program instruction

Syntax

EXECUTE /C **task_num program(**param_list**)**, cycles, step

Function

Begin execution of a control program.

Usage Considerations

A program cannot already be active as the specified program task.

Parameters

/C	Optional qualifier that conditionally attaches the selected robot. The qualifier has an effect only when starting the execution of task 0.
task_num	Real value or expression specifying which program task is to be activated. (For more information on program tasks, see the section Executing Programs in the *eV+ Language User's Guide*.)
program	Name of the program to be executed.
param_list	Optional list of constants, variables, or expressions separated by commas, that must correspond in type and number to the arguments in the .PROGRAM statement for the program specified. If no arguments are required by the program, the list is blank, but the parentheses must be entered. Program parameters may be omitted as desired, using commas to skip omitted parameters. No commas are required if parameters are omitted at the end of the list. Omitted parameters are passed to the called program as undefined and can be detected with the DEFINED real-valued function. Automatic variables (and subroutine arguments) cannot be passed by reference in an EXECUTE instruction. They must be passed by value (see the description of CALL). The parameters are evaluated in the context of the new task that is started (see below).
cycles	Optional real value, variable, or expression (interpreted as an

integer) that specifies the number of program execution cycles to be performed. If omitted, the cycle count is assumed to be 1. For unlimited cycles, specify any negative value. The maximum loop count value allowed is 32,767.

step — Optional real value, variable, or expression (interpreted as an integer) that specifies the step at which program execution is to begin. If omitted, program execution begins at the first executable statement in the program (that is, after the initial blank and comment lines and all the AUTO and LOCAL instructions).

Details

This command initiates execution of the specified control program. The program is executed *cycles* times, starting at the specified program step.

After a program initiates execution of another program, the initiating program can use the STATUS and ERROR real-valued functions to monitor the status of the other program.

The optional /C qualifier has an effect only when starting execution of task 0. When /C is not specified, an EXECUTE instruction for task 0 fails if the robot cannot be attached; attachment requires that the robot be calibrated and that arm power be enabled (or that the DRY.RUN switch is enabled). When /C is specified, an execute instruction for task 0 attempts to attach the robot, but allows execution of task 0 to continue without any indication of error if the robot cannot be attached.

Certain default conditions are assumed whenever program execution is initiated. They are equivalent to the following program instructions:

```
CPON ALWAYS
DURATION 0 ALWAYS
FINE 100 ALWAYS
LOCK 0
MULTIPLE ALWAYS
NULL ALWAYS
OVERLAP ALWAYS
SPEED 100,100 ALWAYS
SELECT ROBOT = 1
```

Also, the robot configuration is saved for subsequent motions.

An execution cycle is terminated when a STOP instruction is executed, a RETURN instruction is executed in the top-level program, or the last defined step of the program is encountered. The value of *cycles* can range from -32,768 to 32,767. The program is executed one time if *cycles* is omitted or has the value 0 or 1. Any negative value for *cycles* causes the program to be executed continuously until a HALT instruction is executed, an error occurs, or you (or another program) aborts execution of the program.

NOTE: Each time an execution cycle is initiated, the execution parameters are reset to their default values. This includes motion speed, robot configuration, and servo modes. However, the robot currently selected is not changed.

If *step* is specified, the program begins execution at that step for the first pass. Successive cycles *always* begin at the first executable step of the program.

All the instruction parameters are evaluated in the context of the *new* task that is started. This can lead to unexpected results when the EXECUTE program instruction is used, and an attempt is made to pass a task-dependent value (for example, the TASK real-valued function). In such a case, if you want the task-dependent value to reflect the *invoking* task, you must assign the task-dependent value to a variable and pass that variable.

Examples

Initiate execution (as task #0) of the program named assembly, with execution to continue indefinitely (that is, until execution is aborted, a HALT instruction is executed, or a run-time error occurs):

```
EXECUTE 0 assembly, -1
```

Initiate execution, with program task #2, of the program named test. The parameter values 1 and 2 are passed to the program.

```
EXECUTE 2 test(1,2)
```

The following program segment shows how an application program can be initiated from another application program (the ABORT and CYCLE.END program instructions are used to make sure the specified program task is not already active):

```
ABORT 3                    ;Abort any program already active
CYCLE.END 3                ;Wait for execution to abort
EXECUTE 3 new.program      ;Start up the new program
```

Related Keywords

ABORT monitor command

ABORT program instruction

CALL program instruction

CYCLE.END monitor command

CYCLE.END program instruction

EXECUTE monitor command

KILL monitor command

KILL program instruction

PRIME monitor command

PROCEED monitor command

RETRY monitor command

SSTEP monitor command

STATUS monitor command

STATUS real-valued function

XSTEP monitorcommand

EXIT program instruction

Syntax

EXIT count

Function

Branch to the statement following the nth nested loop of a control structure.

Usage Considerations

This instruction works with the FOR, WHILE, and DO control structures.

Parameter

count	Optional integer value (expressions and variables are not acceptable) specifying how many nested structures to exit. The default value is 1.

Details

When an EXIT instruction is reached, the control structure is terminated and execution continues at the first instruction following the outermost control structure exited.

Example

If input signal 1001 is set, exit one control structure; if 1002 is set, exit three control structures:

```
27 FOR i = 1 TO 40
28     WHILE ctrl.var DO
29         DO
30             IF SIG(1002) THEN
31                 EXIT 3          ; Jump to step 40
32             END
33             IF SIG(1001) THEN
34                 EXIT            ; Jump to step 37
35             END
36         UNTIL FALSE
37         count = count+1
38     END
39 END
```

Related Keywords

DO program instruction

FOR program instruction

NEXT program instruction

WHILE program instruction

FALSE real-valued function

Syntax

FALSE

Function

Return the value used by eV+ to represent a logical false result.

Usage Considerations

The word "false" cannot be used as a program name or variable name.

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is zero.

Example

The following program loop will execute continuously until the subroutine cycle returns a FALSE value for the real variable **continue**:

```
DO
    CALL cycle(continue)
UNTIL continue == FALSE
```

Related Keywords

OFF real-valued function

TRUE real-valued function

FCLOSE program instruction

Syntax

FCLOSE (logical_unit)

Function

Close the disk file, graphics window, or graphics icon currently open on the specified logical unit.

Usage Considerations

No error is generated if a file or graphics window is not open on the logical unit, although the IOSTAT real-valued function returns an error code.

When a graphics window is closed, the window is not deleted from graphics memory and its stacking and display status are not changed.

Parameter

logical_unit	Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)

Details

After a program has finished accessing a file that has been opened via an FOPEN instruction, the program must close the file by executing an FCLOSE instruction. FCLOSE frees the file for access by the eV+ monitor and other programs. In addition, for files that have been opened for writing, FCLOSE writes out any data still buffered by eV+ and updates the file directory information. Thus, if this operation is not performed, the disk file may not actually contain all of the information written to it.

If a program is finished accessing a graphics window, or needs to reuse its logical unit number, the window can be closed with this instruction. After a window is closed, it can be deleted with an FDELETE instruction or it can be opened again later with an FOPEN instruction.

> **NOTE:** Reopening a window resets all its text and graphics attributes (for example, color, font ID, character path and orientation, texture, logical operation, and enabled events), which must be explicitly reset by the program before attempting output to the window.

An FCLOSE operation is automatically performed on a logical unit when the unit is detached, when the program that issued the FOPEN completes execution, or when a KILL of the program task is performed.

The IOSTAT real-valued function should be used to check for successful completion of a close operation. (The error code for File not opened will be returned if there was no file or window currently open on the specified logical unit.)

Related Keywords

ATTACH program instruction

DETACH program instruction

FOPEN program instruction

FOPENR program instruction

IOSTAT real-valued function

KILL monitor command

KILL program instruction

FCMND program instruction

Syntax

FCMND (logical_unit, command_code) $out_string, $in_string

Function

Generate a device-specific command to the input/output device specified by the logical unit.

Usage Considerations

The logical unit referenced must have been previously attached.

As appropriate, the current default device, unit, and directory path are considered for any disk file specification (see the DEFAULT command).

Parameters

logical_unit	Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)
command_code	Real-valued expression that specifies the command to be executed. (See the explanation of command codes below.)
$out_string	String constant, variable, or expression that is transmitted to the device along with the command code to specify the operation to be performed.
$in_string	Optional string variable. This variable receives any information returned from the device as a result of the command.

Details

This instruction allows a program to generate device-specific command sequences. For example, this instruction can be used to send a command to the disk to delete a file or to rename a file. Since these are maintenance operations, which are not generally performed by eV+ programs, no special-purpose eV+ program instructions exist for performing these operations.

Any error in the specification of this instruction (such as attempting to access an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual operations (such as device not ready) do not halt program execution since these errors can occur in the normal operation of a program. These normal errors can

be detected by using the IOSTAT function after performing the FCMND. In general, it is good practice always to test whether each FCMND operation completed successfully using IOSTAT.

File Command Codes

With the exception of the CLOSE command, a file cannot be open on the logical unit when the FCMND is executed.

6 Rename a file. The $out_string parameter must contain the new name of the file (including any required disk unit and directory path specification). The $in_string variable must contain the old file name.

7 Compress the disk. This command is invalid for local disks.

8 Format the disk. The $out_string parameter must contain the name of the disk unit to format, followed by any required qualifiers. The data contained in $out_string must be identical to that of the argument list of a FORMAT monitor command. On completion, the $in_string variable will contain text indicating how many bad blocks were located.

CAUTION: Formatting a disk erases all the information on the disk.

14 Create a subdirectory. The $out_string parameter must contain the specification of the subdirectory, including an optional unit name if the current default disk unit is not to be accessed. (Refer to the *eV+ Operating System User's Guide* for a description of subdirectory specifications.)

> **NOTE:** Only the final subdirectory in the specified directory path is created by this operation. That is, all the intermediate subdirectories must already exist, and they are not created.

15 Delete a subdirectory. The $out_string parameter must contain the specification of the subdirectory, including an optional unit name if the current default disk unit is not to be accessed. (Refer to the *eV+ Operating System User's Guide* for a description of subdirectory specifications.)

> **NOTE:** Only the final subdirectory in the specified directory path is deleted by this operation. That is, all the intermediate subdirectories must already exist, and they are not deleted.

19 Assert the creation date/time for the file currently open on the specified logical unit. This command can be issued at any time a disk file is opened. Once asserted, when the file is closed, the file's creation date and time are set equal to the specified values rather than the current date and time. Also, if this command is issued when the file is closed, eV+ does not automatically assert the not archived bit. The input string must contain **date** and **time**, where:

date is a 16-bit integer word representing the date in the standard compressed format used by the TIME and $TIME functions.

time is a 16-bit integer word representing the time in the standard compressed format.

This command code applies only to local disk drives.

20 Return the number of unused and total number of KB on a local disk. The returned string is in the form uuuuu/ttttt where uuuuu is the number of unused KB and ttttt is the total number of KB. A file must be open on the drive (with prereads disabled). The open file identifies the disk unit.

21 Read the creation date/time for the file currently open on the specified logical unit. This command can be issued any time after a file has been opened. Normally, this command returns the values that are read from the disk directory at the time the file was opened. However, if an FCMND 19 instruction has been issued to assert file creation date and time, FCMND 21 returns the value set by FCMND 19. The string returned by this command contains **date** and **time** (use INTB to extract the values), where

date is a 16-bit integer word representing the date in the standard compressed format used by the TIME and $TIME functions.

time is a 16-bit integer word representing the time in the standard compressed format.

This command code applies only to local disk drives.

Serial Line Command Codes

102 Clear the type-ahead buffer for a serial line, or clear the event queue for a graphics window. This command, which is recognized only by the serial communication lines and the graphics logical units, does not process any arguments.

106 Read modem control flags. Modem control flags are returned in the first byte of a one-byte string. Bits within that byte show the current state of the modem control lines for the serial port attached to the lun specified in the FCMND instruction. The bits are interpreted as follows (LSB is 1):

Bit	Mask	State of:
1	^H01	Request to Send (RTS)
2	^H02	Data Terminal Ready (DTR)
5	^H10	Input Clear to Send (CTS)
6	^H20	Input Data Carrier Detect (DCD)

DDCMP Command Codes

500 Return information about DDCMP status. (This FCMND is present in all eV+ systems that support DDCMP.) The FCMND reply string may be parsed using INTB and LNGB functions to extract the binary data, as described in the following code. When the instruction

```
FCMND (lun,500) "", $reply
```

is executed, the string variable $reply receives packed binary data regarding the DDCMP line attached on the specified logical unit. Then the functions shown in the following table can be used to extract the data.

DDCMP Status Format

Function	Notes
INTB($reply,1) DDCMP network state (0, 1, or 2)	0 = Line is closed 1 = Line is open but waiting for remote 2 = Line is active
INTB($reply,3)	Not used
LNGB($reply,5)	Local media error count
LNGB($reply,9)	Local timing error count
LNGB($reply,13)	Local format error count
LNGB($reply,17)	Remote media error count
LNGB($reply,21)	Remote timing error count
LNGB($reply,25)	Remote format error count
LNGB($reply,29)	Count of blocks sent
LNGB($reply,33)	Count of blocks received

501 Set DDCMP communication parameters. This command is recognized only by serial communication lines configured for use with the DDCMP protocol. See the *eV+ Language User's Guide* for the details of this command.

TCP Command Codes

600 Initiate a close connection from the TCP server side for the client identified by the handle number **handle** in the instruction FCMND (lun, 600) $INTB(handle). Note, however, that close-connection requests are more commonly initiated by the client side.

601 Initiate a PING command (see the *eV+ Operating System Reference Guide* for details on the PING monitor command). The resulting IOSTAT value is either 1, indicating the client was found on the network, or -562, indicating a network timeout.

DeviceNet Command Codes

761 Used for reading CanBus status. The CanBus is the bus that DeviceNet runs on. When the instruction

```
FCMND (lun,761) "", $input
```

is executed, the string variable $input returns the status shown in the following table.

DeviceNet Status Format

Function	Notes
INTB($input, 1)	CanBus status value (see below)
INTB($input, 3)	Number of bytes transmitted
INTB($input, 5)	Number of acknowledges received
INTB($input, 7)	Number of bytes received
INTB($input, 9)	Number of errors
INTB($input, 11)	Number of bytes lost
INTB($input, 13)	Number of overruns

CanBus status value is a bitmask containing the following bits:

Bit	Meaning When Bit is Nonzero
1	Online
2	Bus warning
3	Bus off
4	Activity detected
5	(reserved)
6	Transmit timeout
7	Receive buffer overrun
8	(reserved)
9	(reserved)
10	(reserved)
11	(reserved)
12	(reserved)
13	Online at 125 KBaud
14	Online at 250 KBaud
15	Online at 500 KBaud
16	Scanner active

762 Used for generic I/O to the DeviceNet scanner. This FCMND reads from the scanner input area. When the instruction

```
FCMND(lun, 762) $INTB(macid)+$INTB(offset)+$INTB
(count), $input
```

is executed, macid> is the MAC ID to read from, offset is the read offset

into the input area (it is device-dependent), and count is the number of bytes to read. The input is returned by the string variable $input. See the Adept MV Controller User's Guide for details on the MACID statement in DeviceNet configuration.

763 Used for generic I/O to the DeviceNet scanner. This FCMND writes to the scanner output area. When the instruction

```
FCMND(lun, 763) $INTB(macid)+$INTB(offset)+$INTB
(count)+$output
```

is executed, macid is the MAC ID to write to, offset is the write offset into the output area (it is device-dependent), and count is the number of bytes to write. The output bytes are contained in the string variable $output. See the Adept MV Controller User's Guide for details on the MACID statement in DeviceNet configuration.

768 Used for determining the DeviceNet status with the following eV+ code:

```
        AUTO lun, macid, status, $error[25], $input

        macid = 1                               ;MacID for this
example

        ; Define the status messages.

        $error[0]  = "Device not in device list"
        $error[1]  = "Device idle (not being scanned)"
        $error[2]  = "Device being scanned"
        $error[3]  = "Device timed-out"
        $error[4]  = "UCMM connection error"
        $error[5]  = "Master/Slave connection set is busy"
        $error[6]  = "Error allocating Master/Slave connection
set"
        $error[7]  = "Invalid vendor id"
        $error[8]  = "Error reading vendor id"
        $error[9]  = "Invalid device type"
        $error[10] = "Error reading device type"
        $error[11] = "Invalid product code"
        $error[12] = "Error reading product code"
        $error[13] = "Invalid I/O input size for connection 1"
        $error[14] = "Error reading I/O input size for
connection 1"
        $error[15] = "Invalid I/O output size for connection
1"
        $error[16] = "Error reading I/O output size for
connection 1"
        $error[17] = "Invalid I/O input size for connection 2"
```

```
        $error[18] = "Error reading I/O input size for
connection 2"
        $error[19] = "Invalid I/O output size for connection
2"
        $error[20] = "Error reading I/O output size for
connection 2"
        $error[21] = "Error setting I/O packet rate for
connection 1"
        $error[22] = "Error setting I/O packet rate for
connection 2"
        $error[23] = "M/S connection set sync fault"

        ; Report the status of DeviceNet.

        ATTACH (lun, 4) "DEVICENET"            ;Attach the
device

        IF IOSTAT(lun) < 0 GOTO 100
        FCMND (lun, 760) $INTB(macid), $input  ;Get the status
        IF IOSTAT(lun) < 0 THEN
            TYPE "No node with MacId", macid
        ELSE
            status = ASC($input)
            TYPE "MacID", macid, " status:  ", $error[status]
        END
        DETACH (lun)                           ;Release the
device
```

Examples

Return modem control bit flags for the serial port attached to logical unit 10:

```
FCMND (10,106), $temp
flags = ASC($temp)
```

Format the disk loaded in drive A in double-sided, double-density format and return the string containing the bad-block count in $bad:

```
FCMND (5, 8) "A:/Q", $bad
```

Specifya DDCMP time-out interval of 2 seconds, with maximums of 20 time-outs and 8 NAK retries.

```
FCMND (lun, 501) $CHR(2)+$CHR(20)+$CHR(8)
```

Check to see if a client is on the network.

```
FCMND (lun, 601) "node_address", $str
```

Related Keywords

ATTACH program instruction

DETACH program instruction

FDELETE monitor command

FDELETE program instruction

FDIRECTORY monitor command

FOPEN program instruction

FRENAME monitor command

IOSTAT real-valued function

MCS program instruction

FCOPY program instruction

Syntax

FCOPY err**, $new_file = $old_file**

Function

Copy the information in an existing disk file to a new disk file.

Parameters

err	Optional parameter, used to return an error.
$new_file	String constant, variable, or expression that specifies the file for the new disk file to be created. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command).
$old_file	String constant, variable, or expression that specifies an existing disk file. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate.

Details

If the new file already exists, or the old file does not exist, an error is reported and no copying takes place. (You cannot overwrite an existing file-the existing file must first be deleted with an FDELETE command.)

If the file to be copied has the special "read-only" attribute, the new file will also have that attribute. Files with the "protected" attribute cannot be copied. (See FDIRECTORY for a

description of file protection attributes.) When a file is copied, the file creation date and time are preserved along with the standard file attributes. The only attribute that is affected is the "archived" bit, which is cleared to indicate that the file is not archived.

In general, a file specification consists of six elements:

1. An optional physical device (for example, DISK>)
2. An optional disk unit (for example, D:)
3. An optional directory path (for example, DEMO\)
4. A file name (for example, NEWFILE)
5. A period character (".")
6. A file extension (for example, V2)

FCOPY can also be used to write a file to a serial line:

```
FCOPY SERIAL:n>="myfile"             ;Global serial line "n"
```

Example

Create a file named "newfile.v2" on disk device "D" that is an exact copy of the existing file named "oldfile.v2" on disk device "D":

```
FCOPY "D:\newfile.v2" = "D:\oldfile.v2"
```

Related Keywords

FCOPY monitor command

DEFAULT monitor command

FRENAME monitor command

FDELETE program instruction

Syntax

FDELETE (logical_unit) object

Function

Delete the specified disk file, the specified graphics window and all its child windows, or the specified graphics icon.

Usage Considerations

The logical unit number must be attached, but no file or window can be currently open on that logical unit.

The window cannot be deleted if it (or any of its child windows) is open as any other logical unit or by any other program task.

Parameters

logical_unit	Real value, variable, or expression (interpreted as an integer) that corresponds to a disk or window logical unit. (See the ATTACH instruction for a description of logical unit numbers.)
object	String constant, variable, or expression specifying the disk file, graphics window, or graphics icon to delete. The error *Nonexistent file* will be reported (via IOSTAT) if the specified object does not exist. For disk files, the string may contain an optional disk unit and an optional directory path, and must contain a file name, a period (.), and a file extension. The current default disk unit and directory path are considered as appropriate (see the DEFAULT command). For graphics windows, the string must fully specify the position in the window tree of the window to be deleted. For graphics icons, the string must specify the name of the icon, followed by /ICON.

Details

If a disk logical unit number is specified, the **object** parameter is interpreted as the specification of a disk file to be deleted. If the deletion fails for any reason (for example, the file does not exist or the disk is protected), an error will be returned via the IOSTAT real-valued function.

NOTE: In order to delete a file from a 3.5 inch diskette, the write-protect slider must be in the position that covers the hole.

If the logical unit number specified is for a graphics window, the **object** parameter is interpreted as the specification of a graphics window or icon to be deleted. When a window is specified, that window *and* all of its child windows are deleted. If any of the window's children cannot be deleted, the specified window is not affected and an error is returned (via the IOSTAT real-valued function). When a window is deleted, it is erased from the display. (A window must be FCLOSEd before it can be FDELETEd.)

When a graphics logical unit is accessed, a *Protection error* message is reported (via IOSTAT) if a system window or icon is specified.

Examples

Delete the disk file defined by the file specification in the string variable $file:

```
FDELETE (5) $file
```

Delete the top-level window named TEST and all of its child windows. The logical unit defined by main must be a graphics logical unit:

```
FDELETE (main) "TEST"
```

Delete the graphics window named ERROR, which is a child of the top-level window named VISION:

```
FDELETE (21) "VISION\ERROR"
```

Delete the graphics icon named BUTTON:

```
FDELETE (20) "BUTTON/ICON"
```

Related Keywords

ATTACH program instruction

FCLOSE program instruction

FDELETE program instruction

FOPEN program instruction

IOSTAT real-valued function

FEMPTY program instruction

Syntax

FEMPTY (logical_unit)

Function

Empty any internal buffers in use for a disk file or a graphics window by writing the buffers to the file or window if necessary.

Usage Considerations

When accessing a file, the file must be open for *random access* on the specified logical unit (see the FOPEN_ instructions).

When accessing a graphics window, this instruction is useful only for a window that is opened in buffered mode. (That is, the /BUFFERED attribute was specified in the FOPEN instruction that opened the window.)

Parameter

logical_unit	Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)

Details

During random-access I/O of a disk file, eV+ writes data to the disk in blocks of 512 bytes (characters). For efficiency, when a record with a length of less than 512 bytes is written using a WRITE instruction, that data is stored in an internal buffer and might not actually be written to the disk until a later time.

When a disk logical unit is referenced, the FEMPTY instruction directs eV+ to write its internal buffer contents immediately to the disk file. That is useful, for example, in applications where data integrity is especially critical (see FOPEN for details on defeating buffering).

When a window logical unit is referenced, the FEMPTY instruction forces all buffered graphics output to be immediately written to the window.

The IOSTAT real-valued function can be used to determine if any error results from an FEMPTY operation.

Examples

Empty the internal output buffer for logical unit 5 and write it to the disk immediately:

```
FEMPTY (5)
```

Empty the internal buffer for graphics logical unit 20 by writing it to the window immediately:

```
FEMPTY (20)
```

Related Keywords

ATTACH program instruction

FOPEN program instruction

FOPEN_ program instruction

IOSTAT real-valued function

WRITE program instruction

FINE program instruction

Syntax

FINE tolerance *ALWAYS*

Function

Enable a high-precision feature of the robot hardware servo.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

If the tolerance parameter is specified, its value becomes the default for any subsequent FINE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified).

This is the default state of the eV+ system. FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The FINE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the FINE instruction causes an error.

Parameters

tolerance	Optional real value, variable, or expression that specifies the percentage of the standard fine tolerances that are used for each joint of the robot attached by the current execution task.
ALWAYS	Optional qualifier that establishes FINE as the default condition. That is, FINE will remain in effect continuously until disabled by a COARSE instruction. If ALWAYS is not specified, the FINE instruction will apply only to the next robot motion.

Details

Enables the high-precision feature in the robot motion servo system so that only small errors in the final positions of the robot joints are permitted at the ends of motions. This produces high-accuracy motions but increases cycle times since the settling time at the end of each motion is increased.

If the tolerance parameter is specified, the new setting takes effect at the start of the next motion. Also, the value becomes the default for any subsequent FINE instruction executed

during the current execution cycle (regardless of whether or not ALWAYS is specified). Changing the FINE tolerance does not affect the COARSE tolerance.

If the tolerance parameter is omitted, the most recent setting (for the attached robot) is used. The default setting is restored to 100 percent when program execution begins, or a new execution cycle starts (assuming that the robot is attached to the program).

Examples

Enable the high-precision feature only for the next motion:

```
FINE
```

Enable the high-tolerance feature for the next motion, with the tolerance settings changed to 50% of the standard tolerance for each joint (that is, a tighter tolerance):

```
FINE 50
```

Enable the high-tolerance feature until it is explicitly disabled:

```
FINE ALWAYS
```

Related Keywords

COARSE program instruction

CONFIG real-valued function

DELAY.IN.TOL program instruction

NONULL program instruction

NULL program instruction

SELECT program instruction

SELECT real-valued function

FLIP program instruction

Syntax

FLIP

Function

Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a flip configuration, this instruction is ignored by the robot.

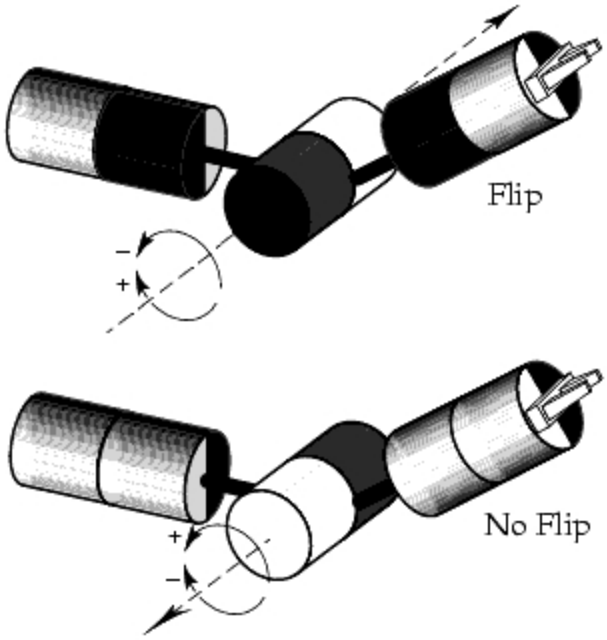
The FLIP instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the FLIP instruction causes an error.

Details

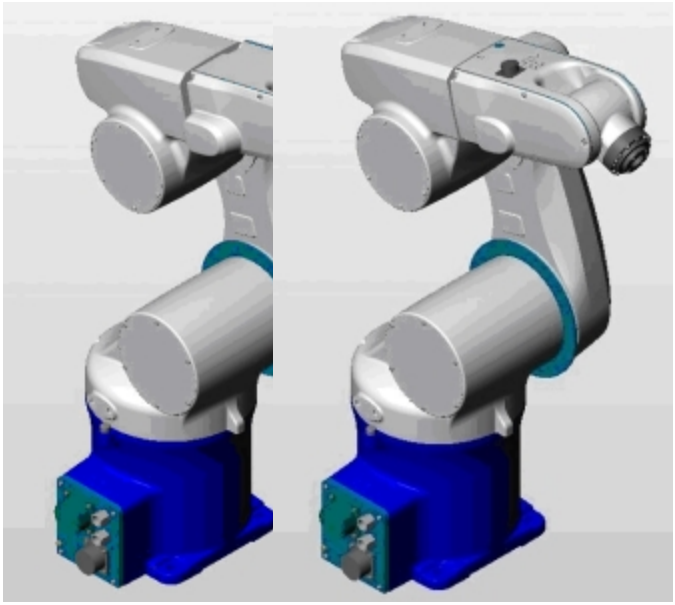
Asserting a FLIP configuration forces the wrist joint to have a negative rotation (top robot in FLIP/NOFLIP). Asserting a NOFLIP configuration forces a wrist joint to have a positive rotation (bottom robot in FLIP/NOFLIP). Wrist joint angles are expressed as ±180°.

> **NOTE:** Robots can change configuration only during joint-interpolated moves.



FLIP/NOFLIP

The following figures illustrate FLIP versus NOFLIP configurations of a Viper s650 robot.



FLIP / NOFLIP Example on Viper s650 Robot

Example

The following eV+ code snippet demonstrates the use of the FLIP and NOFLIP program instructions:

```
FLIP          ;Request change in robot configuration during next
motion
MOVE loc_a    ;Move to loc_a transformation with FLIP
configuration

NOFLIP        ;Request change in robot configuration during next
motion
MOVE loc_a    ;Move to loc_a transformation with NOFLIP
configuration
```

Related Keywords

CONFIG real-valued function

NOFLIP program instruction

SELECT program instruction

SELECT real-valued function

FLTB real-valued function

Syntax

FLTB ($string, first_char**)**

Function

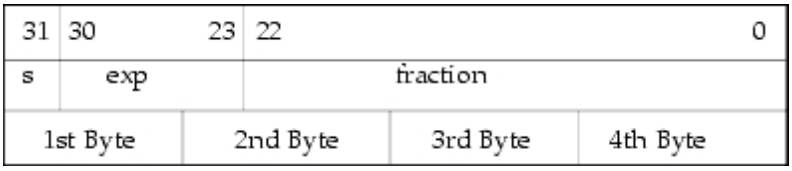
Return the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.

Parameters

$string	String expression that contains the four bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the four bytes in the string. If first_char is omitted or has a value of 0 or 1, the first four bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second, third, fourth, and fifth bytes are extracted. An error is generated if first_char specifies four bytes that are beyond the end of the input string.

Details

Four sequential bytes of the given string are interpreted as being a single-precision (32-bit) floating-point number in the IEEE standard format. This 32-bit field is interpreted as follows:



where

s is the sign bit, s = 0 for positive, s = 1 for negative.

exp is the binary exponent, biased by -127.

fraction is a binary fraction with an implied 1 to the left of the binary point.

For 0 < exp < 255, the value of a floating-point number is:

$$-1^{s} * (1.\text{fraction}) * 2^{\text{exp}-127}$$

For exp = 0, the value is zero; for exp = 255, an overflow error exists.

The main use of this function is to convert a binary floating-point number from an input data record to a value that can be used internally by eV+.

Examples

```
FLTB($CHR(^H3F)+$CHR(^H80)+$CHR(0)+$CHR(0))        ;Returns 1.0
FLTB($CHR(^HC0)+$CHR(^H40)+$CHR(0)+$CHR(0))        ;Returns -3.0
```

Related Keywords

ASC real-valued function

DBLB real-valued function

$DBLB string function

$FLTB string function

INTB real-valued function

LNGB real-valued function

$LNGB string function

TRANSB transformation function

VAL real-valued function

$FLTB string function

Syntax

$FLTB (value)

Function

Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.

Parameter

value	Real-valued expression, the value of which is converted to its IEEE floating-point binary representation.

Details

A real value is converted to its binary representation using the IEEE single-precision standard floating-point format. This 32-bit value is packed as four successive 8-bit characters in a string. See the FLTB real-valued function for a more detailed description of IEEE floating-point format.

The main use of this function is to convert a real value to its binary representation in an output record of a data file.

Example

```
$FLTB(1.215)
;Returns a character string equivalent to:
$CHR(^H3F)+$CHR(^H9B)+$CHR(^H85)+$CHR(^H1F)
```

Related Keywords

$CHR string function

DBLB real-valued function

$DBLB string function

FLTB real-valued function

$INTB string function

LNGB real-valued function

$LNGB string function

$TRANSB string function

FOPEN program instruction

Syntax

FOPEN (logical_unit, mode**) attribute_list**

Function

Create and open a new graphics window or TCP connection, or open an existing graphics window for subsequent input or output.

Open a graphics icon for definition.

Usage Considerations

The logical unit must be attached before an open operation will succeed.

The TCP mode of operation of the FOPEN instruction applies only to Adept controllers with the license for the AdeptTCP/IP Protocol Access.

Parameters

logical_unit	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number assigned to the window or TCP device. (See the ATTACH instruction for a description of unit numbers.)
mode	Optional expression that applies only to TCP logical units and selects the type of TCP connection: 0 = Client mode, 16 = Server mode.
attribute_list	List of string constants, variables, and expressions; real values, variables, and expressions; and format specifiers used to assign a name to the window and to define some of the characteristics of the window. When opening a TCP connection in server mode, this string defines the characteristics of the server. When opening a connection in client mode, the string defines the name of the server in addition to characteristics of the connection. The attribute list (which is processed like an output specification for the TYPE instruction) is used to compose a single string that is passed to the window manager or TCP driver. The string must begin with the name of the window and can optionally contain keyword attributes that define characteristics of the window. The string must not exceed 512 characters.

NOTE: A eV+ string literal or string variable cannot exceed 128 characters. In order to create an attribute list longer than 128 characters you must concatenate multiple strings.

The attribute list can consist of one or more components separated by commas. Each component can be expressed in any of the following ways:

1. A string constant, variable, or expression.
2. A real-valued constant, variable, or expression, which is evaluated to determine a value to be used in the control string.
3. A format-control specifier, which determines the format of information in the control string.

Details

Using FOPEN With TCP

A TCP/IP *connection* can be opened in either *server mode* or *client mode*. In server mode, one or more clients (depending on the value assigned to /CLIENTS) are allowed to connect to the server for subsequent communication.

To establish a client-server connection, the client must know the port number for the server. For this reason, when using the FOPEN instruction for opening a server connection, the port is explicitly defined using the /LOCAL_PORT attribute. Note that the server does **not** need to know the port number used by the client.

Port numbers 0 through 255 are used by standard TCP application packages. For example, FTP uses ports 20 and 21. By convention, if you are writing your own custom protocol, use a port number greater than 255.

The following table shows valid TCP attributes for the FOPEN instruction.

FOPEN TCP Attributes

Attribute:	/CLIENTS
Explanation:	Defines the number of client connections allowable on a server. If omitted, a single client connection is assumed. The maximum number of client connections is 31.
Attribute:	/LOCAL_PORT

Explanation:	Defines the local port number for the connection. If omitted, a local port number is automatically assigned.
Attribute:	/REMOTE_PORT
Explanation:	Defines the port number of a server to which a client connection is to be made. This *must* be provided when establishing a client connection.

Examples

Set up a TCP server with local port #260 to accept 5 client connections:

```
FOPEN (lun, 16) "/LOCAL_PORT 260 /CLIENTS 5"
```

Set up a TCP client connection that connects to port number 260 on the server called server1:

```
FOPEN (lun, 0) "server1 /REMOTE_PORT 260"
```

Related Keywords

ATTACH program instruction

DETACH program instruction

FCLOSE program instruction

FDELETE program instruction

FEMPTY program instruction

FSET program instruction

IOSTAT real-valued function

FOPEN_ program instruction

Syntax

FOPEN_ (lun, record_len, mode**) file_spec**

Function

Open a disk file for read-only, read-write, read-write-append, or read-directory, as indicated by the last letter of the instruction name.

The forms of FOPEN_ are:

- FOPENA
- FOPEND
- FOPENR
- FOPENW

See the Details section for descriptions of each instruction.

Usage Considerations

A logical unit must be attached before an open operation will succeed.

No more than 60 disk files and 160 network files can be open by the entire system at any time. That includes files opened by all of the program tasks and by the system monitor (for example, for an FCOPY command).

Parameters

lun	Real-valued expression defining the logical unit number of the disk device to be accessed. (See the ATTACH instruction for a description of unit numbers.)
record_len	Optional real-valued expression defining the length of records to be read and written. If the record length is omitted or is zero, variable-length records are processed. In this case, random access of records cannot be done. If the record length is nonzero, it specifies the length (in characters) of fixed-length records to be processed. Random access is allowed with fixed-length records.
mode	Optional real-valued expression defining how read access is to be done. The value specified is interpreted as a sequence of bit flags as

detailed below. (All bits are assumed to be clear if no mode value is specified.)

Bit 1 (LSB) Disable prereads (mask value = 1)

If this bit is clear, eV+ will read a record as soon as the file is opened (a preread) and after each READ instruction in anticipation of subsequent READ requests. If this bit is set, no such prereads are performed.

Bit 2 Enable random access (mask value = 2)

If this bit is clear, the file will be accessed sequentially. That is, records are read or written in the order they occur in the file.

If this bit is set, the file is accessed using random access (which is allowed only for disk files with fixed-length records). In random-access mode, the record-number parameter in the READ or WRITE instruction specifies which record is accessed.

Bit 4 Force disk write (mask value = 8)

If set for a disk file being opened for write access, the physical disk is written every time a record is written. In addition, the directory or file allocation information is updated with each write. This mode is equivalent to (but faster than) closing the file after every write. It is *much slower* than normal buffered mode, but it guarantees that information that is written will not be lost due to a system crash or power failure. This mode is intended primarily for use with log files that are left opened over an extended period of time and intermittently updated. For these types of files, the additional (significant) overhead of this mode is not as important as the benefit.

file_spec String constant, variable, or expression specifying the file to be opened. The string may contain an optional disk unit and an optional directory path, and must contain a file name, a period (.), and a file extension. (For FOPEND, the file name and extension are optional, and both can contain wildcard characters-see below.)

The current default disk unit and directory path are considered as appropriate (see Using Directories for additional information on disk units and directory paths.)

Details

This instruction opens a disk file so that input/output (I/O) operations can be performed. When the I/O operations are complete, the file should be closed using an FCLOSE or DETACH instruction.

FOPENA — Opens a file for read-write-append access. If the specified file does not already exist, the file is created.

If the file already exists, no error occurs, and the file position is set to the end of the file. Write operations then append to the existing file.

FOPEND — Opens a disk directory for reading. The file name and extension in the file_spec parameter are used to prepare a file name template for use when read operations are later performed. Those read operations return only records from the disk directory file that match the file name template. Any attempt to write to the directory file causes an error. (For information on the format of directory records, see the section Accessing the Disk Directories in the *eV+ Language User's Guide*.)

The file name and extension can include wildcard characters (asterisks, *). A wildcard character within a file name or extension indicates that any character should be accepted in that position. A wildcard character at the end of a file name or extension indicates that any trailing characters are acceptable. A wildcard character in place of a file name (or extension) indicates that any name (or extension) is acceptable. Omission of the file name, the period, and the file extension is equivalent to specifying *.*. Omission of the period and file extension is equivalent to specifying a wildcard extension.

FOPENR — Opens a file for read-only access. If the file does not already exist, an error occurs. Any attempt to write to the file causes an error.

FOPENW — Opens a file for read-write access. If the file already exists, an error occurs.

Any error in the specification of this instruction (such as attempting to access an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual operations (such as device not ready) do not halt program execution since these errors can occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function.

Example

```
FOPENR (5) "data.dat"
```

Open the file named data.dat on the default device for read-only access with variable-length records (record length omitted). Since the mode parameter is omitted, prereads will occur and the records will be accessed sequentially (which is required for variable-length records).

```
FOPENW (5, 32, 3) "D:x.d"
```

Open the file named x.d on the device D for read-write access using fixed-length records of 32 characters each. The mode value 3 has both bits 1 and 2 set; thus, prereads are to be disabled and random access is to be used.

```
FOPEND (5) "*.dat"
```

Open the current default directory to find all the files with the extension DAT.

Related Keywords

ATTACH program instruction

DETACH program instruction

FCLOSE program instruction

FOPEN_ program instruction

IOSTAT real-valued function

FOR program instruction

Syntax

FOR loop_var = initial TO final *STEP* increment

Function

Execute a group of program instructions a certain number of times.

Usage Considerations

An END instruction must be included in a program to match every FOR.

Parameters

loop_var	Real valued variable that is initialized when the FOR instruction is executed and is incremented each time the loop is executed (cannot be a specified value or expression).
initial	Real value that determines the value of the loop variable the first time the loop is executed.
final	Real value that establishes the value to be compared to the loop variable to determine when the loop should be terminated.
increment	Optional real-value that establishes the value to be added to the loop variable every time the loop is executed. If omitted, the increment defaults to one, and the keyword STEP may also be omitted.

Details

The instructions between the FOR statement and the matching END statement are executed repeatedly, and **loop_var** is changed each time by the value of increment.

The processing of this structure is as follows:

1. When the FOR statement is first entered, set **loop_var** to the **initial** value.
2. Determine the values of the increment and **final** parameters.
3. Compare the value of **final** to the value of **loop_var**:
    - If increment is positive and **loop_var** is greater than **final**, skip to item 7 below.

- If increment is negative and **loop_var** is less than (that is, more negative than) **final**, skip to item 7 below.

4. Execute the group of instructions following the FOR statement.
5. When the END step is reached, add the value of increment to the loop variable.
6. Go back to item 3 above.
7. Continue program execution at the first instruction after the END statement. **loop_var** retains the value it had at the time of the test in item 3 above.

Note that the group of instructions in the FOR structure may not be executed at all if the test in item 3 fails the first time.

The values of **initial**, increment, and **final** when the FOR statement is first executed determine how many times the group of instructions are executed. Any changes to the values of these parameters within the FOR loop have no effect on the processing of the FOR structure.

Changes to the loop variable within the loop affect the operation of the loop and should normally not be done.

> **NOTE:** If initial, final, or increment are not integer values, rounding in the floating point computations may cause the loop to be executed more or fewer timer than expected.

Example

The following example sets all elements of a 10x10 array to 0:

```
FOR i = 1 TO 10
    FOR j = 1 TO 10
        array[i,j] = 0
    END
END
```

Related Keywords

DO program instruction

EXIT program instruction

NEXT program instruction

WHILE program instruction

FRACT real-valued function

Syntax

FRACT (value)

Function

Return the fractional part of the argument.

Parameter

value	Real-valued expression whose fractional part is returned by this function.

Details

The fractional part of a real value is the portion to the right of the decimal point (when the value is written without the use of scientific notation).

The value returned has the same sign as the function argument.

Examples

```
FRACT(0.123)         ;Returns 0.123
FRACT(-5.462)        ;Returns -0.462
FRACT(1.3125E+2)     ;Returns 0.25 (1.3125E+2 = 131.25)
```

Related Keyword

INT real-valued function

FRAME transformation function

Syntax

FRAME (location_1, location_2, location_3, location_4)

Function

Return a transformation value defined by four positions.

Parameters

location_1	Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame.
location_2	Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame.
location_3	Transformation, compound transformation, or a transformation-valued function whose position is used to define the Y axis of the computed frame.
location_4	Transformation, compound transformation, or a transformation-valued function whose position is returned as the position of the computed frame transformation.

Details

While the robot can be used to define an X, Y, Z position very accurately, it is often difficult to define precisely an orientation. For applications such as palletizing, the FRAME function is very useful for accurately defining a base transformation whose position and orientation are determined by four positions. This function returns a transformation value that is computed as follows:

1. Its origin is at the point defined by **location_4**.
2. Its positive X axis is parallel to the line passing through the points defined by **location_1** and **location_2**, in the direction from **location_1**to **location_2**.
3. Its X-Y plane is parallel to the plane that contains the points defined by **location_1**, **location_2**, and **location_3**.
4. Its positive Y direction is from the computed X axes (as defined above), toward the point defined by **location_3**.

Example

The following instruction defines the transformation base.frame to have the same X, Y, Z position as origin, its X axis parallel to the line from center to x, and its Y axis approximately in the same direction as the line from center to y.

```
SET base.frame = FRAME(center, x, y, origin)
```

Related Keyword

TRANS transformation function

FREE real-valued function

Syntax

FREE (memory, select**)**

Function

Return the amount of unused free memory storage space.

Parameters

memory Optional real value, variable, or expression (interpreted as an integer) that specifies which portion of system memory is to be examined, as shown below. The value zero is assumed if the parameter is omitted.

memory	Memory examined
0	Program memory
1	Obsolete
2	Obsolete

select Optional real value, variable, or expression (interpreted as an integer) that specifies what information about the memory is to be returned, as shown below. The value zero is assumed if the parameter is omitted.

select	Information returned
0	Percentage of memory available
1	Available memory, in KB (1024 bytes)
2	Obsolete

NOTE: If both parameters are omitted, the parentheses must still be included.

Details

This function returns the information displayed by the FREE command. Unlike the FREE command, however, this function returns only one value, determined by the values specified for the *memory* and *select* parameters.

Related Keyword

FREE monitor command

FSEEK program instruction

Syntax

FSEEK (logical_unit, record_number**)**

Function

Position a file open for random access and initiate a read operation on the specified record.

Usage Considerations

A file must be open for random access on the specified logical unit (see the FOPEN_ instruction).

For efficiency in most applications, the file should be opened in no preread mode.

Parameters

logical_unit	Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)
record_number	Optional real-valued expression that specifies the record to read for file-oriented devices opened in random-access mode. If omitted, the record following the one last read is assumed.

Details

When a file is open for random access, system performance can be improved by overlapping the time required for disk file access with processing of the current data. By using the FSEEK instruction, an application program can initiate a disk seek and possible read operation immediately after a READ instruction is processed but before processing the data.

Any error in the specification of this instruction (such as referencing an invalid unit) causes a program error and halts program execution. However, errors associated with performing the actual seek operation (such as end of file or device not ready) do not halt program execution since these errors may occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function after performing the subsequent READ operation. In general, it is good practice always to test whether each file operation completed successfully by testing the value from IOSTAT.

Example

```
; Seek record number 130 in the file open on logical unit 5:
    FSEEK (5, 130)
```

Related Keywords

ATTACH program instruction

FOPEN program instruction

IOSTAT real-valued function

READ program instruction

FSET program instruction

Syntax

FSET (logical_unit) attribute_list

Function

Set or modify attributes of a serial line or a network device.

Usage Considerations

If a window has been referenced, it must have been opened already with an FOPEN instruction. If a serial line is referenced, it must have been attached already with an ATTACH instruction.

The use of this instruction withnetwork devices applies only to systemswith the appropriate license(s).

As with all eV+ I/O instructions, the IOSTAT real-valued function should be used after each FSET instruction to determine the success of the FSET request.

Parameters

logical_unit	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
attribute_list	List of string constants, variables, and expressions; real values, variables, and expressions; and format specifiers used to define the characteristics of the window. See the description of the FOPEN instruction for detailed information on this parameter.

Details

Using FSET With Serial Lines

The following specifications can be used as arguments to directly ATTACH a serial line:

SERIAL:n
Local serial line n on the local controller

The keywords listed in the following table may appear in the keyword list string.

FSET Serial-Line Attributes

Attribute	Argument	Description
/PARITY	NONE	No parity generation
	EVEN	Use even parity
	ODD	Use odd parity
/STOP_BITS	1 or 2	Use 1 or 2 stop bits per byte
/BYTE_LENGTH	7 or 8	Use 7 or 8 bits per byte
/FLOW	NONE	No flow control
	XON_XOFF	Detect and generate XON/XOFF (turn off modem)
	MODEM	Use modem control RTS/CTS (turn off XON_XOFF).
/SPEED	110, 300, 600, 1200, 2400, 4800, 7200, 9600, 19200, 38400, 57600, 115200	Select the indicated baud rate.

Using FSET With TCP

The following network devices may be referenced with the FSET> instruction:

TCP Transmission Control Protocol

You can use the attributes listed in the following table when accessing these devices with the FSET instruction:

FSET Attributes for Networks

Attribute	Description
/ADDRESS	IP address. (Applies only to the TCP device.)
/NODE	Node name.

You may define new nodes on the network using the FSET program instruction to access a logical unit that has been attached to the TCP device. The string used with the FSET

instruction has the same format as that used with the NODE statement in the eV+ configuration file (see the later example).

Examples

Serial

The following example attaches serial line 2 and sets the baud rate to 38400:

```
ATTACH (slun, 4) "SERIAL:2"
FSET (slun) "/SPEED 38400"
```

Network

Define a new node called SERVER2 with the IP address 172.16.200.102:

```
ATTACH (lun, 4) "TCP"
FSET (lun) "/NODE 'SERVER2' /ADDRESS 172 16 200 102"
```

Related Keywords

FOPEN program instruction

IOSTAT real-valued function

GETC real-valued function

Syntax

GETC (lun, mode)

Function

Return the next character (byte) from a device or input record on the specified logical unit.

Usage Considerations

The logical unit must be attached by the program for normal, variable-length record input/output.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of the unit numbers.)

mode Real value, variable, or expression (interpreted as an integer) that specifies the mode of the read operation. Currently, the mode is used only for the terminal and serial I/O logical units. The value is interpreted as a sequence of bit flags as detailed below. (All bits are assumed to be clear if no mode value is specified.)

Bit 1 (LSB) Disable waiting for input (mask value = 1)

If this bit is clear, program execution is suspended until the next byte is received. If the bit is set and no bytes are available, the function immediately returns the error code for *No data received* (-526).

> **NOTE:** A -526 error may be returned by the first no-wait GETC even if there are bytes queued.

Bit 2 Disable echo (mask value = 2)

If this bit is clear, input from the terminal is echoed back to the source. If the bit is set, characters are not echoed back to the source. (This bit is ignored for the serial lines.)

Details

The next byte from the device is returned. When reading from a record-oriented device such as the system terminal or a disk file, the carriage-return and line-feed characters at the end

of records are also returned. When the end of a disk file is reached, a Ctrl+Z character (26 decimal) is returned.

When reading from the terminal, GETC will return the next character entered at the keyboard. All control characters will be read, except Ctrl+S, Ctrl+Q, Ctrl+O, and Ctrl+W, which will have their normal terminal control functions.

When reading from the serial line, GETC will return the next data byte immediately, unmodified. (Note that if the serial line is configured to recognize Ctrl+S and Ctrl+Q automatically as control characters, then those characters are not returned by the GETC function.)

Normally, the byte value returned is in the range 0 to 255 (decimal). If an input error occurs, a negative error code number is returned. The meanings of the error codes are listed in the section System Messages.

Example

The following program segment reads characters from a disk file until a comma (,) character, a control character, or an I/O error is encountered. The characters are appended to the string variable $field. (The disk file must have already been opened for accessing variable-length records.)

```
$field = ""
c = GETC(5)
WHILE (c > ^H1F) AND (c <> ',) DO
    $field = $field+$CHR(c)
    c = GETC(5)
END
IF c < 0 THEN
    TYPE $ERROR(c)
    HALT
END
```

Related Keywords

ATTACH program instruction

READ program instruction

GET.EVENT real-valued function

Syntax

GET.EVENT (task**)**

Function

Return events that are set for the specified task.

Usage Considerations

Do not confuse GET.EVENT with the GETEVENT program instruction, which returns information from a graphics window or the terminal.

Parameter

task Optional real value, variable, or expression (interpreted as an integer) that specifies the task for which events are to be returned. The valid range is -1 to 6, or -1 to 27, inclusive. If the parameter is omitted, or has the value -1, the current task is referenced.

> **NOTE:** The basic system allows 7 tasks (0–6). The eV+ Extensions option allows 28 tasks (0–27).

Details

The events are returned in a value that should be interpreted as a sequence of bit flags, as detailed below.

Bit 1 (LSB) I/O Completion (mask value = 1)

This bit being set indicates that a system input/output operation has completed.

See the descriptions of SET.EVENT and WAIT.EVENT for more details.

Related Keywords

CLEAR.EVENT program instruction

SET.EVENT program instruction

WAIT.EVENT program instruction

GLOBAL program instruction

Syntax

GLOBAL type **variable,** ..., variable

Function

Declare a variable to be global and specify the type of the variable.

GLOBAL statements must appear before any executable statement in the program.

Parameters

type	Optional parameter specifying the type of a variable. The acceptable types are:	
	LOC	Location variable (transformation, precision point, belt)
	REAL	Single-precision real variable
	DOUBLE	Double-precision real variable
		See the Details section for the default type.
variable	Variable name (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array. If the type parameter is specified, all the variables must match the specified type. Array variables must not have their indexes specified.	

Details

Variables that are not declared to be AUTO or LOCAL are GLOBAL by default. Undeclared scalar variables default to double precision.

Thus, double-precision and location global variables do not need to be declared.

Global variables can be seen by any program that does not declare a LOCAL or AUTO variable of the same name. Thus, if **program_a** declares var1 to be a GLOBAL variable and **program_b** declares var1 to be AUTO, **program_b** cannot use or alter GLOBAL var1. A new copy of variable var1 that is specific to **program_b** is created each time **program_b** executes.

Examples

```
GLOBAL $str_1, $str_2, x      ;create 2 string and 1 untyped
variable
GLOBAL LOC #ppoint_1          ;create 1 global precision point
variable
GLOBAL var_1, var_2           ;create 2 double prec. reals
```

Related Keywords

AUTO program instruction

LOCAL program instruction

GOTO program instruction

Syntax

GOTO label

Function

Perform an unconditional branch to the program step identified by the given label.

Parameter

label	Label of the program step to which execution is to branch. Step labels are integer values that range in value from 0 to 65535.

Details

This instruction causes program execution to jump to the line that contains the specified step label. Note that a step **label** is different from a line number. Line numbers are the numbers automatically assigned by the eV+ program editors to assist the editing process. Step labels must be explicitly entered on program lines where appropriate.

Modern, structured programming considers GOTO statements to be poor programming practice. Adept suggests you use one of the other control structures in place of GOTO statements.

Example

The following program segment asks you to enter a number from 1 to 100. If the number input is not in that range, the GOTO 10 instruction at line number 27 causes execution to jump to step **label** 10 (at line number 23).

```
21  ; Get a number from the user
22
23  10 PROMPT "Enter a number from 1 to 100: ", number
24
25  IF (number < 1) OR (number > 100) THEN
26      TYPE /B, /C1, *Invalid response*, /C1
27      GOTO 10
28  END
```

Related Keywords

DO program instruction

EXIT program instruction

FOR program instruction

IF ... THEN program instruction

NEXT program instruction

WHILE program instruction

HALT program instruction

Syntax

HALT

Function

Stop program execution and do not allow the program to be resumed.

Usage Considerations

The PROCEED command cannot be used to resume program execution after a HALT instruction causes the program to halt.

HALT forces an FCLOSE and/or DETACH on the disk and serial communication logical units as required.

Details

Causes a BREAK and then terminates execution of the application program regardless of any program loops remaining to be completed (see the EXECUTE command and instruction). The message (HALTED) is displayed.

After termination by a HALT instruction, program execution cannot be resumed with a PROCEED or RETRY command.

Related Keywords

PAUSE program instruction

RETURN program instruction

STOP program instruction

HAND real-valued function

Syntax

HAND

Function

Return the current hand opening.

Usage Considerations

The HAND function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the HAND function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word "hand" cannot be used as a program name or variable name.

Details

This function returns 0 if the hand is closed or 1 if the hand is opened or relaxed.

Related Keywords

CLOSE program instruction

CLOSEI program instruction

OPEN program instruction

OPENI program instruction

RELAX program instruction

RELAXI program instruction

SELECT program instruction

SELECT real-valued function

HAND.TIME system parameter

Syntax

... HAND.TIME

Function

Establish the duration of the motion delay that occurs during OPENI, CLOSEI, and RELAXI instructions.

Usage Considerations

The current value of the HAND.TIME parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the HAND.TIME parameter can be modified only with the PARAMETER monitor command or program instruction.

The parameter name can be abbreviated.

If the eV+ system is controlling more than one robot, the HAND.TIME parameter controls the hand operation times for all the robots.

Details

The OPENI, CLOSEI, and RELAXI instructions are used to operate the hand after the robot has stopped moving. The HAND.TIME parameter determines the time allotted to the hand actuation before the next robot motion can be initiated.

The value for this parameter is interpreted as the number of seconds to delay. It can range from 0 to 10^{18}. Because of the way eV+ generates time delays, the HAND.TIME parameter is internally rounded to the nearest multiple of 0.016 seconds.

This parameter is set to 0.05 seconds when the eV+ system is initialized.

Example

Set the hand operation delay time to 0.5 seconds:

```
PARAMETER HAND.TIME = 0.5
```

Related Keywords

CLOSEI program instruction

OPENI program instruction

RELAXI program instruction

PARAMETER monitor command

PARAMETER program instruction

PARAMETER real-valued function

HERE program instruction

Syntax

HERE location_var

Function

Set the value of a transformation or precision-point variable equal to the current robot location.

Usage Considerations

The HERE instruction returns information for the robot selected by the task executing the instruction.

If the eV+ system is not configured to control a robot, executing the HERE instruction does not generate an error because of the absence of a robot. However, the location value returned may not be meaningful.

The word "here" cannot be used as a program name or variable name.

Parameter

location_var	Transformation, precision point, or compound transformation that ends with a transformation variable.

Details

This instruction sets the value of a transformation or precision-point variable equal to the current robot location.

Normally, the robot location is determined by reading the instantaneous values of the joint encoders. However, if the robot has either backlash or linearity compensation enabled, the commanded robot location is used.

If the **location_var** is a compound transformation, only the right-most transformation is defined. Its value is set equal to the current robot location relative to the reference frame determined by the other transformations. An error message results if any of the other transformations are not already defined.

Examples

Set the transformation part equal to the current robot location:

```
HERE part
```

Assign the current location of the robot to the precision point #part:

```
HERE #part
```

Related Keywords

HERE monitor command

HERE transformation function

SELECT program instruction

SELECT real-valued function

SET program instruction

HERE transformation function

Syntax

HERE

Function

Return a transformation value that represents the current location of the robot tool point.

Usage Considerations

The current location is obtained by reading the instantaneous value of the joint encoders so that it represents the actual location of the robot.

The HERE function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the HERE function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word "here" cannot be used as the name of a program or variable.

Example

Calculate the distance between the current robot location and the location the robot is currently moving to:

```
dist = DISTANCE(HERE, DEST)
```

Related Keywords

DEST transformation function

HERE monitor command

HERE program instruction

SELECT program instruction

SELECT real-valued function

ID real-valued function

Syntax

ID (component, device, board**)**

Function

Return values that identify the configuration of the current system.

Parameters

component	Real value, variable, or expression (interpreted as an integer) whose value determines which component of identification information is returned.
device	Optional real value, variable, or expression (interpreted as an integer) whose value selects the device to be identified. Device #1 (the basic system) is assumed if this parameter is omitted.
board	Optional integer specifying the CPU of interest when the device parameter value equals 4. Board #1 (the main CPU) is assumed if this parameter is omitted.

Details

The ID function enables a program to access the information displayed by the ID monitor command. The values of the components are the same as the fields displayed by that command.

The function returns the value 0 for devices that do not exist. Device numbers that do not exist return the value 0. For valid devices, an *Invalid argument* error message is reported if the requested component is not valid.

The following table describes the type of information returned when the **device** parameter is set at a specified value. To see the acceptable values for the component parameter and the type of information returned for each device value, click on the **device** link in the table below.

To obtain information on...	Set the device parameter to...
The basic system	device = 1 (This is the default value.)
The pendant	device = 2 returns information about the manual control pendant.
Robot	device = 8 or device = 10+r returns information

To obtain information on...	Set the device parameter to...
	about the robot.
CPU and board info	device = 4 returns information about the CPU and processor board (if the board parameter is specified).

component	Result of ID (component, 1)
1	Model designation of the system controller
2	Serial number of the system controller
3	Version number of the eV+ software in use. This is the internal version that is incremented for each software release. It also differentiates V+ from eV+ (V+ ends at version 17.x; eV+ begins at version 20.x). To get the external product version number, which is displayed by the ID Monitor Command, use ID(14,1).
4	Revision number of the eV+ software in use
5	First option word for the eV+ system (*)
6	Second option word for the eV+ system (*)
7	Size of the system program memory (in kilobytes, K [1 K = 1024 8-bit bytes])
8	Not used.
9	Controller product-type value
10	Returns the SmartController base board revision code.
11	Controller hardware configuration. This field should be interpreted as a bit field. For details on the information returned, see the section Controller Hardware Configuration returned by ID(11,1), below.
12	Returns the first and second parts of the security ID (shown as "aaaa-bbbb") below. (The value returned needs to be displayed in

component	Result of ID (component, 1)
	hexadecimal format to look the same as "aaaa-bbbb".) Security ID:aaaa-bbbb-cccc
13	Returns the third part of the security ID (shown as "cccc") below. (The value returned needs to be displayed in hexadecimal format to look the same as "cccc".) Security ID: `aaaa-bbbb-cccc`
14	External version number of the eV+ software in use. Also, see ID (3,1).
*The system option words are described in ID Option Words on page 1.	

Controller Hardware Configuration returned by ID(11,1)

The value for the controller hardware configuration [returned by ID(11,1)] should be interpreted as a bit field, as follows:

Bit 3 - Emulator (mask = 4)

This bit is set if the system is an emulator (e.g. it runs on PC with virtual robot instead of running on a SmartController-EX controlling real robots).

Pendant

Device number 2 refers to the pendant. The **component** parameter must have the value 1. The value returned is the version number of the pendant software.

CPU and Board Configuration

Device number 4 refers to the system CPUs. (If the indicated board does not exist, all values are returned as -1.)

component	Result of ID (component, 4, board)
5	CPU type: 8 = SmartController EX

<table>
<tr><th>component</th><th>Result of ID (component, 4, board)</th></tr>
<tr><td>6</td><td>Bit field indicating which software modules are active on the board. Bit definitions are provided in the following table.
<table>
<tr><th rowspan="2">Bit #</th><th colspan="2">Mask Value</th><th rowspan="2">Interpretation When Bit Set</th></tr>
<tr><th>Decimal</th><th>Hexadecimal</th></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Processor is running the eV+ Operating System</td></tr>
<tr><td>2</td><td>2</td><td>2</td><td>Obsolete</td></tr>
<tr><td>3</td><td>4</td><td>4</td><td>Processor is running the Servo software</td></tr>
<tr><td>4-16</td><td></td><td></td><td>Reserved for future use (currently zero)</td></tr>
</table></td></tr>
</table>

Robot and Encoder Configuration

Device number 8 returns information for the currently selected robot. The acceptable values for the **component** parameter are the same as for a specified robot. (See the following table.)

Device number 10 refers to the external encoders connected to the robot controller. The acceptable values for the **component** parameter are the same as for a robot.

Device numbers 11, 12, ... refer to robot number 1, 2, ..., respectively, for each robot connected to the controller. That is, a device number equal to 10+*r* refers to robot number *r*, which can range from 1 to the value returned by the function SELECT(ROBOT, -1). The number of the robot that is currently selected can be obtained with the function SELECT (ROBOT).The acceptable values for the **component** parameter, and the corresponding values returned, are listed in the following table.

component	Result of ID (component, 10+r)
1	Model designation of the robot

component	Result of ID (component, 10+r)
2	Serial number of the robot
3	Number of motors configured for the robot This normally is equal to the number of configured joints (see component 7).
4	Value interpreted as bit flags for the robot joints that are enabled: bit 1 for joint 1, and so on. (The value is zero if the robot does not have joints that can be disabled selectively. For example, this value is defined for the X/Y/Z/Theta robot but is zero for the 4/5-axis SCARA module.)
5	Robot control-module identification number
6	Obsolete
7	Number of robot joints configured for use
8	Robot option word (*)
9	Robot product-type value
10	Obsolete
11	Second robot option word (*)
12	Information on the robot module Currently, only bit 1 (mask 1) is defined. If set, this bit indicates that the specified robot is an Adept robot.
13	Returns the safety level for the robot. Possible values are: 0 = No special safety level 1 = Configured as Category 1 Robot System per ISO 10218 and EN954 3 = Configured as Category 3 Robot System per ISO 10218 and EN954
14	Returns whether the axis mask is editable or not

*The robot option words are described in the Robot Option Words topic

Related Keywords

ID monitor command

$ID string function

SELECT monitor command

SELECT program instruction

SELECT real-valued function

$ID string function

Syntax

$ID (select)

Function

Return the system ID string.

Parameter

select Integer specifying the ID information to return. It may be:

Integer	Description
-1	Returns the system edit message.
-2	Returns the edit letter and issue number for the eV+ system.
-3	Returns the vision edit message string.
-4	Returns the servo edit message string.

Details

This function returns a string that identifies the release edition and date of the requested system software component.

Related Keywords

ID monitor command

ID real-valued function

IDENTICAL real-valued function

Syntax

IDENTICAL (location, location)

Function

Determine whether two location values are exactly the same.

Parameter

location Transformation value that defines one of the locations of interest. This can be a function, a variable, or a compound transformation.

Details

This function returns the value TRUE if the positional and rotational components of the two specified locations are *exactly* the same. Even a single-bit difference in any of the components results in the value FALSE being returned.

Example

The statement

```
x = IDENTICAL(base.1:loc,part)
```

sets the value of the real variable x to TRUE if the value of loc relative to the base.1 frame is exactly the same as the value stored in the variable part.

Related Keyword

DISTANCE real-valued function

IF logical_expr THEN program instruction

Syntax

IF logical_expr *THEN*

first steps

ELSE

second steps

END

Function

Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.

Usage Considerations

There must be a matching END statement for every IF... THEN in a program.

Parameters

logical_expr	Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (zero).
first_steps	Optional group of program instructions that are executed only if the value of the logical expression is TRUE (nonzero).
second_steps	Optional group of program instructions that are executed only if the value of the logical expression is FALSE (zero). The ELSE statement may be omitted if there are no steps in this group.

Details

This control structure provides a means for conditionally executing one of two groups of instructions. In detail, it is processed as follows:

1. **logical_expr** is evaluated. If the result is FALSE (zero), skip to item 4 below.
2. The first group of instruction steps is executed.
3. Skip to item 5 below.

4. If there is an ELSE step, the second group of instruction steps is executed.
5. Program execution continues at the first step after the END step.

The ELSE and END steps must be on lines by themselves as shown.

There are no restrictions on the instructions that can be in either group in the structure. Thus, nested IF structures can be used.

Examples

Consider the following segment of a eV+ program. If the value of row is greater than 5, the expression row > 5 will be TRUE (-1.0), so step 22 is executed and 24 is not executed. Otherwise, step 22 is not executed, but step 24 is executed:

```
21  IF row > 5 THEN
22      spacing = 10
23  ELSE
24      spacing = 20
25  END
```

The next program segment determines whether the variable input.signal has been defined. If it has, the program checks the signal indicated by the value of input.signal and types different messages depending on its setting. Note that the outer IF does not include an ELSE clause:

```
71  IF DEFINED(input.signal) THEN
72      IF SIG(input.signal) THEN
73          TYPE "The input signal is ON"
74      ELSE
75          TYPE "The input signal is OFF"
76      END
77  END
```

Refer to the DEFINED function for details on testing nonreal arguments.

Related Keywords

CASE program instruction

DEFINED real-valued function

ELSE program instruction

IF... GOTO program instruction

IF logical_expr GOTO program instruction

Syntax

IF logical_expr GOTO label

Function

Branch to the specified step label if the value of the logical expression is TRUE (nonzero).

Usage Considerations

In general, it is a better programming practice to use the IF ... THEN control structure rather than this instruction.

Parameters

logical_expr	Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (zero).
label	Program step label of a step in the current program.

Details

If the value of the expression is nonzero, program execution branches and begins executing the statement with a label matching the one specified. If the value of the expression is zero, the next instruction is executed as usual.

If the specified statement label is not defined, the program is not executable. Any attempt to branch to an undefined label is identified when the program editor is exited and when the program is loaded into memory from a disk file.

Example

The most common use for IF...GOTO is as an exit-on-error instruction. The following code checks each I/O operation and branches to a label whenever an I/O error occurs:

```
        ATTACH(dlun, 4) "DISK"
        IF IOSTAT(dlun) < 0 GOTO 100
        FOPENW(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
            ...
        FCLOSE(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
        DETACH(dlun)
    100 IF IOSTAT(dlun) < 0 THEN
            TYPE $ERROR(IOSTAT(dlun))
        END
```

Related Keywords

GOTO program instruction

IF ... THEN program instruction

IGNORE program instruction

Syntax

IGNORE signal

Function

Cancel the effect of a REACT or REACTI instruction.

Usage Considerations

Only digital I/O signals that are installed and configured as inputs are available for reaction monitoring.

The IGNORE instruction must be executed by the same program task that initiated the REACT or REACTI instruction.

Parameter

signal	Digital input signal number in the range 1001 to 1012, an internal signal in the range 2001 to 2008.

Details

Disables continuous monitoring of the specified signal, canceling the effect of the last REACT or REACTI for this signal.

Example

Stop monitoring of the digital input or soft signal identified by the value of test.

```
IGNORE test
```

Related Keywords

LOCK program instruction

REACT program instruction

REACTI program instruction

INRANGE real-valued function

Syntax

INRANGE (location)

Function

Return a value that indicates whether a location can be reached by the robot and, if not, why not.

Usage Considerations

The INRANGE function returns information for the robot selected by the task executing the function.

Parameter

location Transformation function, variable, or compound that specifies a desired position and orientation for the robot tool tip.

Details

The function returns a value that indicates whether or not the given location can be reached by the robot. The value zero indicates that the specified location can be reached.

If the location cannot be reached, the returned value is a coded binary number that identifies the reason. A bit equal to 1 in the value indicates that the corresponding robot constraint would be violated, as shown in the table below:

	Mask Value		
Bit #	**Hex**	**Decimal**	**Indication if bit set**
1	1	1	Joint or motor 1 is limiting
2	2	2	Joint or motor 2 is limiting
3	4	4	Joint or motor 3 is limiting
4	8	8	Joint or motor 4 is limiting
5	10	16	Joint or motor 5 is limiting
6	20	32	Joint or motor 6 is limiting

	Mask Value		
Bit #	**Hex**	**Decimal**	**Indication if bit set**
7	40	64	Joint or motor 7 is limiting
8	80	128	Joint or motor 8 is limiting
9	100	256	Joint or motor 9 is limiting
10	200	512	Joint or motor 10 is limiting
11	400	1024	Joint or motor 11 is limiting
12	800	2048	Joint or motor 12 is limiting
13	1000	4096	Collision detected
14	2000	8192	Location is too close in
15	4000	16384	Location is too far out
16	8000	32768	Motor is limiting, rather than joint (see below)

If the motion system is configured to return motor-limit as well as joint-limit errors, bit 16 indicates whether a joint or motor would limit motion to location. If bit 16 is set, all the joints passed their limit checks, and the indicated motor is limiting. Otherwise, the indicated joint is limiting.

The mask values indicated above can be used with the BAND operator to determine if a corresponding bit is set.

Example

Returns the value zero if the robot can reach the location defined by the compound transformation *pallet:hole*.

```
INRANGE(pallet:hole)
```

If both joints 2 and 3 would prevent the motion from being made, the value returned would be 6.

Related Keyword

SELECT program instruction

SELECT real-valued function

INSTALL program instruction

Syntax

INSTALL password, op

Function

Install or remove software options available to Adept systems.

Usage Considerations

You must have received the authorization password from Adept. INSTALL can be run only on CPU #1 in multiple CPU systems.

Parameters

password	String expression that contains a 15-character value assigned by Adept.
op	Optional integer indicating the desired operation: 0 = install option (default) 1 = remove option

Details

When you purchase additional software options from Adept, the software is delivered with a software license and authorization password that enables the software for a particular controller. If the option is not enabled, the software does not load correctly.

The password is keyed both to the software option and the serial number of your controller. The password cannot be used on any controller other than the one for which you purchased the software option.

INT real-valued function

Syntax

INT (value)

Function

Return the integer part of the value.

Parameter

value Real-valued expression whose integer part is returned by this function.

Details

Returns the portion of the value parameter to the left of the decimal point (when the value is written without the use of scientific notation).

The value is not rounded before dropping the fraction.

The sign of the value parameter is preserved unless the result is zero.

Examples

```
INT(0.123)                      ;Returns 0.0

INT(10.8)                       ;Returns 10.0

INT(-5.462)                     ;Returns -5.0

INT(1.3125E+2)                  ;Returns 131.0

INT(cost)                       ;Returns the value of "cost",
                                ;truncated to an integer.

INT(cost+0.5*SIGN(cost))        ;Returns the value of "cost",
rounded
                                ;to the nearest integer. (The SIGN
                                ;function needs to be included to
                                ;correctly round negative values of
                                ;"cost".)
```

Related Keyword

FRACT real-valued function

INTB real-valued function

Syntax

INTB ($string, first_char**)**

Function

Return the value of two bytes of a string interpreted as a signed 16-bit binary integer.

Parameters

$string	String expression that contains the two bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the two bytes in the string. If first_char is omitted or has a value of 0 or 1, the first two bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second byte contains bits 9 to 16 and the third byte contains bits 1 to 8. An error is generated if first_char specifies a byte pair that is beyond the end of the input string.

Details

Two sequential bytes of a string are interpreted as being a 2's-complement 16-bit signed binary integer. The first byte contains bits 9 to 16, and the second byte contains bits 1 to 8.

The main use of this function is to convert binary numbers from an input data record to values that can be used internally by eV+.

The expression

```
value = INTB($string, first_char)
```

is equivalent to the following instruction sequence:

```
value = ASC($string,first_char)*256 + ASC($string,first_char+1)
IF value > 32767 THEN
    value = value-65536
END
```

To compute an unsigned integer, use: INTB($string) BAND ^HFFFF.

Examples

```
INTB($CHR(10)+$CHR(5))          ;Returns the value 2565
```

```
INTB($CHR(255)+$CHR(255))        ;Returns the value -1
```

Related Keywords

ASC real-valued function

DBLB real-valued function

FLTB real-valued function

$INTB string function

LNGB real-valued function

VAL real-valued function

$INTB string function

Syntax

$INTB (value)

Function

Return a 2-byte string containing the binary representation of a 16-bit integer.

Parameter

value	Real-valued expression, the value of which is converted to its binary representation.

Details

The integer part of a real value is converted into its binary representation and the low 16 bits of that binary representation are packed into a string as two 8-bit characters. Bits 9-16 are packed first, followed by bits 1-8.

This function is equivalent to:

```
$CHR(INT(value/256) BAND ^HFF) + $CHR(INT(value) BAND ^HFF)
```

The main use of this function is to convert integers to binary representation within an output record of a data file.

Example

```
$INTB(65*256+67)     ;Returns the character string "AC".
```

Related Keywords

$CHR string function

$DBLB string function

$FLTB string function

INTB real-valued function

$LNGB string function

INVERSE transformation function

Syntax

INVERSE (transformation)

Function

Return the transformation value that is the mathematical inverse of the given transformation value.

Parameter

transformation Transformation-valued expression.

Details

Mathematically, the value from this function is a transformation such that the value of the compound transformation shown below is the identity transformation (or NULL).

```
INVERSE(trans):trans
```

Stated another way, consider a transformation x that defines the location of object A relative to object B. Then INVERSE(x) is the transformation that defines the location of object B relative to A.

Example

Consider the case where the location part_1 is known in robot coordinates, and you want to find the location hole_1 with respect to part_1. We can use the compound expression:

```
part_1:hole_1
```

to represent the position of hole_1 in robot coordinates.

Suppose we move the robot to hole_1 and use the HERE command to define hole_pos as the position of hole_1 in robot coordinates. In other words, we want to find hole_1, knowing the values of part_1 and hole_pos, and knowing that:

```
part_1:hole_1 is equal to hole_pos
```

We can then use the INVERSE function to determine hole_1 with the instruction:

```
SET hole_1 = INVERSE(part_1):hole_pos
```

Note that the SET instruction can be used without explicit use of INVERSE by using a compound transformation on the left-hand side, with identical results. That is, the instruction defines hole_1.

```
SET part_1:hole_1 = hole_pos
```

Related Keywords

HERE program instruction

SET program instruction

IOSTAT real-valued function

Syntax

IOSTAT (lun, mode**)**

Function

Return status information for the last input/output operation for a device associated with a logical unit.

Usage Considerations

IOSTAT returns information only for the most recent operation. If more than one operation is performed, the status should be checked after each one.

Parameters

lun Real-valued expression whose integer value is the logical unit number for the I/O device of interest. (See the description of ATTACH for information on the logical unit numbers recognized by the eV+ system and how logical units are associated with I/O devices.)

mode Optional expression that selects the type of I/O status to be returned for the specified logical unit. The following table shows the effects of the various mode values. (If the mode value is omitted, the value zero is assumed.)

Mode	Value returned by IOSTAT
0	Status of the last complete I/O operation
1	Status of a pending preread request
2	Size in bytes of the last file opened or of the last record read[1]
3	Status of any outstanding write request

[1] When sequential-access mode is being used, the byte count returned by IOSTAT(...,2) includes the carriage-return and line-feed characters at the end of each record.

Details

Unlike most eV+ instructions, I/O instructions do not force the program to stop when an error is detected. Instead, the error status is stored internally for access with the IOSTAT function. This feature allows the program to interpret and possibly recover from many I/O errors.

When reading a file of unknown length, IOSTAT is the only method to determine when the end of the file is reached.

The value returned for modes 0, 1, and 3 is one of the following:

IOSTAT Value Returned on EOF	Description
1	Normal success; for mode 3 this value indicates that no write request is outstanding.
0	Operation not yet complete
< 0	Standard eV+ error number. See System Messages for a description of standard eV+ error numbers.

Examples

Try to open a file for reading, and make sure the file exists. If the file does exist, record its size (in bytes).

```
ATTACH (dlun, 4) "DISK"
FOPENR (dlun) "RECORD.DAT"
IF IOSTAT(dlun) < 0 THEN
    TYPE "Error opening file"
    HALT
END
file.size = IOSTAT(dlun,2)
```

Read and display records until the end of the file is reached.

```
ieeof = -504                    ;End-of-file error code
READ (dlun) $record
WHILE IOSTAT(dlun) > 0 DO
    TYPE $record
    READ (dlun) $record
END
IF IOSTAT(dlun) == ieeof THEN
    TYPE "Normal end of file"
ELSE
    TYPE /B, "I/O error ", $ERROR(IOSTAT(dlun))
END
```

```
FCLOSE (dlun)
DETACH (dlun)
```

In the following example a TCP server program segment performs a no-wait read and then checks the status to determine whether a client connection or disconnection was made.

```
ATTACH (lun,4) "TCP"
IF IOSTAT (lun) < 0 THEN
    TYPE "Attach error: ", $ERROR(IOSTAT(lun))
END
no_wait = 1
READ (lun, handle, no_wait) $in.str
status = IOSTAT(lun)
CASE status OF
  VALUE 1:                      ;Data received
    TYPE "Data received.  Handle =", handle
  VALUE 100:                    ;New connection opened
    TYPE "New connection established.  Handle =", handle
  VALUE 101:                    ;Connection closed
    TYPE "Connection closed.  Handle =", handle
  VALUE -526:                   ;No data received
    WAIT
  ANY                           ;Some other error
    TYPE "Error during READ:  ", $ERROR(status)
    GOTO 100
END
```

Related Keywords

ATTACH program instruction

FCLOSE program instruction

FCMND program instruction

FEMPTY program instruction

FOPEN program instruction

FSEEK program instruction

READ program instruction

WRITE program instruction

IPS keyword

Syntax

SPEED value IPS *ALWAYS*

Function

Specify the units for a SPEED instruction as inches per second.

> **NOTE:** To specify speed in millimeters per second, use the MMPS conversion factor.

Usage Considerations

IPS can be used only as a parameter for a SPEED program instruction.

The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.

Speeds specified with the IPS parameter apply to straight-line motions. Joint-interpolated motions do not maintain the specified tool speed.

Details

IPS is an optional parameter for the SPEED program instruction, which specifies the units to be used for the speed value. That is, when IPS is specified in a SPEED instruction, the speed value is interpreted as inches/second (for straight-line motions).

See the description of the SPEED program instruction for further details on setting motion speeds with the IPS conversion factor.

Example

Set the robot tool tip speed to 20 inches/second for the next straight-line robot motion (assuming the monitor speed is set to 100):

```
SPEED 20 IPS
```

Related Keywords

MMPS keyword

SPEED program instruction

JHERE program instruction

Syntax

JHERE variable1, ..., variablen

Function

Records the current robot joint positions in real or double-precision variables. This instruction supports MicroeV+.

Parameters

variable1 A real or double-precision variable to receive the position of joint 1.

variablen A real or double-precision variable to receive the position of joint n.

Details

You can specify a maximum of 12 variables. The variables can be array elements with index expressions.

You can omit variables from the list as desired. The following example records the joint-3 and joint-5 positions respectively:

```
JHERE , , j3,,j5
```

If more variables are defined than there are joints for a robot, the extra variables are zero.

Related Keyword

JMOVE program instruction

JMOVE program instruction

Syntax

JMOVE expression1,...,expressionn

Function

Moves all robot joints to positions described by a list of joint values. The robot performs a coordinated motion in joint-interpolated mode. This instruction is supported by MicroeV+.

Parameters

expression1	An optional expression for the joint-1 value.
expressionn	An optional expression for the nth joint value.

> **NOTE:** You must specify at least one expression (joint), in order to move the robot.

Details

You can specify a maximum of 12 expressions.

If an expression is omitted, that joint is not moved. The following example moves only joint 1 and joint 3.

```
JMOVE j1,,j3
```

If more expressions are specified than there are joints for a robot, the extra expressions are ignored.

Related Keyword

JHERE program instruction

JOG program instruction

Syntax

JOG (status) **robot, mode, axis, speed,** location, appro_dist

Function

Moves ("jogs") the specified joint of the robot, or moves the robot tool along the specified Cartesian direction. Each time JOG is executed, the robot moves for up to 300 ms.

Usage Considerations

The specified robot cannot be attached by any other task when using a mode other than COMP. Otherwise, the error message *Robot interlocked* is generated. The robot can be attached by the current program, but it does not need to be attached. If the robot is not attached when the JOG instruction is executed, remember to attach the robot after the JOG instruction before executing any other motion instructions.

After the robot is moved with the JOG instruction, the system is left in MANUAL mode (i.e., as though a manual mode had been selected on the pendant). JOG mode 5 (or the pendant) can be used to restore COMP mode. Otherwise, an error *COMP mode disabled* will be returned when a task attempts to attach the robot.

If a joint is out of range, the JOG instruction can be used to bring the joint back into range. See the Details section for more information.

Parameters

status — An optional status variable (returns 1 for success; otherwise, contains a eV+ error code)

robot — Specifies the robot number.

mode — Specifies the jog mode, as follows:

-1	Keep-alive mode. Continues the previous instruction for another 300 ms.
1	Free joint mode. A positive speed will put the specified joint(s) in Free mode. A negative speed will put the specified joint(s) out of Free mode.
2	Individual joint control.

3	World coordinates control.
4	Tool coordinates control.
5	Restore COMP mode.
6	unused.
7	Jog toward the specified location using the specified speed.
8	Jog toward alignment of the robot tool-Z axis with the nearest World axis.
9	Cartesian control relative to a frame defined by the specified location.

(See the **Details** for information about errors associated with the modes.)

axis Specifies the joint number or Cartesian coordinate (X=1, Y=2, ...), depending on the specified jog mode (see above), for the desired motion.

This parameter is ignored for modes 7 and 8, but a value must always be specified.

speed Specifies the speed and direction of the motion. This is interpreted as a percentage of the speed in manual mode. Values above 100 are interpreted as 100%, values below -100 are interpreted as -100%.

If Free mode is specified, a positive speed will put the given joint in free mode and a negative speed will put the joint out of free mode.

location Optional parameter that specifies a transformation value. This parameter is ignored (and can be omitted) for all modes except 7 and 9.

appro_dist Optional real-valued expression that specifies the distance along the robot tool Z axis between the specified location and the actual desired destination.

A positive distance sets the tool back (negative tool-Z) from the specified location; a negative distance offsets the tool forward (positive tool-Z). This parameter is used only for mode 7.

Details

When the status variable is supplied, and there is an error, the JOG instruction does not cause program execution to stop. The error is simply returned in the STATUS variable.

Each time the JOG instruction is executed, the robot moves for up to 300 ms. Another JOG can be executed before the previous motion is completed. In fact, for extended smooth motion, subsequent JOG instructions should be executed within 300 ms of the previous JOG instruction. The keep-alive mode can be used for that purpose. The keep-alive mode will have no effect after the timeout of 300 ms; it has an effect only before the robot stops.

The following error conditions can be reported when the instruction is processed:

- Mode 1: The error *Illegal joint number* (-609) is returned if FREE mode is not permitted for the specified joint.
- Mode 2: The error *Joint control of robot not possible* (-938) is returned if the robot does not support joint control.
- Modes 3, 4, 8, 9: The error *Cartesian control of robot not possible* (-635) is returned if the robot does not support Cartesian control.
- Mode 7: If the location cannot be reached, the motion stops at the limit of possible motion and the error *Location out of range* (-610) is returned when the motion stops. If any other motion error occurs during the motion (e.g., an obstacle is encountered), the associated error is reported.
- Modes 7 and 9: The error *Missing argument* (-454) is returned if a location is not specified. For mode 7, a straight-line motion is performed toward the specified location if the location is specified with a transformation. A joint-interpolated motion is performed if the location is specified with a precision point. However, if the robot does not permit the type of motion associated with how the location is specified (e.g., the Adept Quattro robot does not permit joint-interpolated motion), the motion is performed in the manner that is permitted by the robot.

When a robot joint is out-of-range, it can be driven into range in either of these ways:

- Go into MAN mode on the pendant, and manually control the joint.
- Put the pendant in COMP mode, and use the JOG instruction to move the joint back into range. (JOG is allowed only in pendant COMP mode.)

> **NOTE:** Use of COMP mode when a joint is out of range is very restricted. All motion instructions (except JOG) return a *Position out of range* error in that situation. In addition, JOG can move the joint only in the direction that moves the joint back into range..

Examples

The following are some examples of proper use of the JOG instruction:

```
JOG 1, 2, 3, -10          ;JOG joint 3 in negative direction in
                          ; JOINT mode
JOG 1, 3, 1, 10           ;JOG X-axis in WORLD mode
JOG 1, 4, 2, 10           ;JOG Y-axis in TOOL mode
JOG 1, 7, 1, 10,loc1      ;JOG toward loc1
JOG 1, 7, 1, 10,loc1, 50 ;JOG toward 50 mm above loc1
```

Related Keywords

JMOVE program instruction

JOG monitor command

MOVE program instruction

KILL program instruction

Syntax

KILL task_number

Function

Clear a program execution stack and detach any I/O devices that are attached.

Usage Considerations

KILL cannot be used while the specified program task is executing.

KILL has no effect if the specified task execution stack is empty.

Parameter

task_number	Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be cleared. (See below for the default. See the *eV+ Language User's Guide* for information on tasks.)

Details

This operation clears the selected program execution stack, closes any open files, and detaches any I/O devices that may have been left attached by abnormal program termination.

This situation can occur if a program executes a PAUSE instruction or is terminated by an ABORT command or instruction, or an error condition, while an I/O device is attached or a file is open. If a limited-access I/O device (such as the serial I/O device) is left attached, no other program task can use that device until it is detached.

The KILL instruction always accesses task #0 if the task number is omitted.

Related Keywords

ABORT monitor command

ABORT program instruction

EXECUTE monitor command

EXECUTE program instruction

STATUS monitor command

LAST real-valued function

Syntax

LAST (array_name[])

Function

Return the highest index used for an array (dimension).

Usage Considerations

If an automatic variable is referenced (see the AUTO instruction), this function returns the index specified in the AUTO statement that declared this array, regardless of which elements have been assigned values.

Parameter

array_name[]	Name of the array to be tested. Any type of eV+ array variable can be specified: real-value, location, string, or belt. At least one array index must be omitted (see below).

Details

This function can be used to determine which elements of an array have already been defined. For one-dimension arrays (for example, part[ ]), this function returns the largest array index for which an element is defined. (See the first example below.)

For multiple-dimension arrays (for example, $names[,]), this function returns the largest array index for which an element is defined for the (left-most) dimension that is omitted from the array specification. (See the second example below.) There cannot be an index specified to the right of an omitted index.

Note that the value returned by this function is an index, not an array element. Furthermore, the value is not a count of the array elements that are defined-it is the largest index for which an array element is defined.

The value -1 is returned if the array does not have any elements defined for the requested dimension. That is, -1 is returned if any of the following situations occur:

- The array does not exist.
- The array has more or fewer dimensions than the number indicated in the function call. (For example, LAST(a[ ]) will return -1 if the array **a** has two dimensions.)
- The specified dimension in a multiple-dimension array has not been defined at all. (For example, LAST(a[20,]) returns -1 if LAST(a[,]) returns 19. That is, no elements a [20,i] exist.)

The error *Illegal array index* results if there is not at least one blank index in the array specification supplied to this function, or if there is an index specified to the right of a blank index.

Examples

If the array part[ ] has all its elements defined from part[0] through part[10], the following example returns the value 10 (not 11, the number of elements defined).

```
LAST(part[])
```

If the given two-dimension array has elements [2,0], [2,3], and [2,5] defined, the following example returns the value 5 (regardless of the status of elements [i,j] for i other than 2).

```
LAST($names[2,])
```

LATCH transformation function

Syntax

LATCH (select)

Function

Return a transformation value representing the location of the robot at the occurrence of the last external trigger or Stop on Digital Signal.

Usage Considerations

LATCH(0) returns information for the robot selected by the task executing the function. If the eV+ system is not configured to control a robot, use of the LATCH(0) function does not generate an error because of the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select Optional integer, expression, or real variable specifying:

0 Robot position latch of currently selected robot (default)
n Robot position latch of robot n

Details

LATCH() returns a transformation value that represents the location of the robot when the last external trigger occurred or the last Stop On Digital Signal occurred. The LATCHED real-valued function should be used to determined when an external trigger has occurred and a valid location has been recorded.

Operation of the external trigger can be configured from the V+ System Configuration Editor in the Adept ACE software. For details, see the *Adept ACE User's Guide*.with Adept ACE.

The DEVICE real-valued function may be used to read the latched value of an external encoder

Related Keywords

DEVICE real-valued function

LATCHED real-valued function

#PLATCH precision-point function

LATCHED real-valued function

Syntax

LATCHED (select)

Function

Return the status of the position latch and which input triggered it or the status of the Stop On Digital Signal.

Parameter

select	Integer, expression, or real variable that determines whether any latches have occurred since the last time the function was executed:
0	Returns latch information for currently selected robot
-n (< 0)	Returns latch information for belt n
+n (> 0)	Returns latch information for robot n

Return Value	0 if no latch has been detected N if position latch on the rising edge of input N was detected -N if position latch on the falling edge of input N was detected

NOTE: N represents any digital input signal on the controller, from 1001 to 1012

Details

This function returns a nonzero value if a position latch or the Stop on Digital Signal event occurred (and thus the robot location or belt-encoder position has been latched) since the LATCHED function was last used. Otherwise, the function returns the value FALSE. When this function returns a nonzero value, the data for the latch event is made available for retrieval by the following functions:

- DEVICE Returns position of external encoder
- BELT Returns position of external encoder
- LATCH Returns robot location as a transformation
- #PLATCH Returns robot location as a precision point

NOTE: After one or multiple nonzero values are returned by this function and the latch buffer is empty, subsequent use of the function returns the value FALSE until the next occurrence of a latch trigger.

Operation of the position latch can be configured from the V+ System Configuration Editor in the Adept ACE software. For details, see the *Adept ACE User's Guide*.

Related Keywords

DEVICE real-valued function

BELT real-valued function

LATCH transformation function

#PLATCH precision-point function

LEFTY program instruction

Syntax

LEFTY

Function

Request a change in the robot configuration during the next motion so that the first two links of a SCARA robot resemble a human's left arm.

Usage Considerations

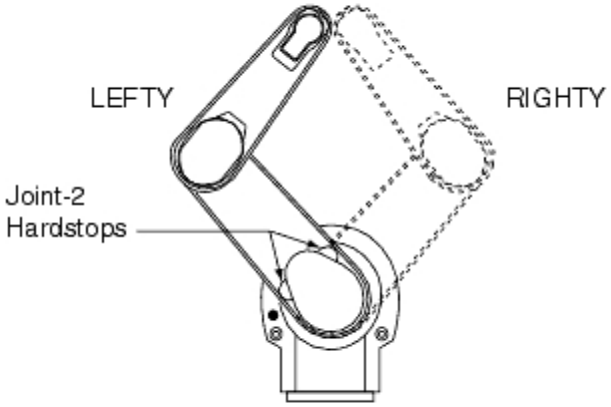
Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a left-handed configuration, this instruction is ignored by the robot.

The LEFTY instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the LEFTY instruction causes an error.

The following figure shows the LEFTY/RIGHTY configurations (top view of robot).



LEFTY/RIGHTY

Related Keywords

CONFIG real-valued function

RIGHTY program instruction

SELECT program instruction

SELECT real-valued function

LEN real-valued function

Syntax

LEN (string)

Function

Return the number of characters in the given string.

Parameter

string	String constant, variable, or expression whose length is to be computed.

Example

Return the number of characters in the string $str:

```
$str = "Hello"
str.len = LEN($str)
```

LNGB real-valued function

Syntax

LNGB ($string, first_char**)**

Function

Return the value of four bytes of a string interpreted as a signed 32-bit binary integer.

Usage Considerations

Since single-precision numbers are stored internally with only 24 bits of significance, input values that contain more than 24 significant bits are converted with some loss in precision.

Double-precision numbers are stored with 32 bits of significance with the MSB being the sign bit. Doubles are converted with no loss of precision.

Parameters

$string	String constant, variable, or expression that contains the four bytes to be converted.
first_char	Optional real value, variable, or expression (interpreted as an integer) that specifies the position of the first of the four bytes in the string. An error results if first_char specifies a series of four bytes that goes beyond the end of the input string. If first_char is omitted or has the value 0 or 1, the first four bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte (see below).

Details

Four sequential characters (bytes) of a string are interpreted as being a 2's-complement 32-bit signed binary integer. The first of the four bytes contains bits 25 to 32 of the integer, the second of the four bytes contains bits 17 to 24, etc.

For example, if first_char has the value 9, then the ninth character (byte) in the input string contains bits 25 to 32 of the integer, the tenth byte of the string contains bits 17 to 24, and so forth.

The main use of this function is to convert binary numbers from an input data record to values that can be used internally by eV+.

Example

Return the value 65541.

```
LNGB($INTB(1)+$INTB(5))
```

Related Keywords

ASC real-valued function

DBLB real-valued function

FLTB real-valued function

INTB real-valued function

$LNGB string function

TRANSB transformation function

VAL real-valued function

$LNGB string function

Syntax

$LNGB (value)

Function

Return a 4-byte string containing the binary representation of a 32-bit integer.

Usage Considerations

Real values are rounded and any fractional part is lost. Values must be in the range ^H7FFFFFFF to -^H80000000

Parameter

value	Real value, variable, or expression whose value is to be converted to its binary representation.

Details

The integer part of a real value is converted into its binary representation; the low 32-bits of that binary representation are packed into a string as four 8-bit characters. Bits 25 to 32 are packed into the first byte, followed by bits 17 to 24 in the second byte, and so forth.

The main use of this function is to convert integer values to binary representation within an output record of a data file.

The operation performed by this function is equivalent to the following expression:

```
$CHR(INT(value/^H1000000) BAND ^HFF)
+ $CHR(INT(value/^H10000) BAND ^HFF)
+ $CHR(INT(value/^H100) BAND ^HFF)
+ $CHR(INT(value) BAND ^HFF)
```

Example

Returns the value $INTB(67)+$INTB(12345).

```
$LNGB(67*65536+12345)
```

Related Keywords

$CHR string function

$FLTB string function

$INTB string function

LNGB real-valued function

$TRANSB string function

LOCAL program instruction

Syntax

LOCAL type **variable, ...,** variable

Function

Declare permanent variables that are defined only within the current program.

Usage Considerations

Subroutines can be called simultaneously by multiple program tasks and recursively by a single task. Local and global variables can be corrupted if such calls occur inadvertently. Thus, the use of automatic variables in place of local variables is recommended.

LOCAL statements must appear before any executable statement in the program.

If a variable is listed in a LOCAL statement, any global variable with the same name cannot be accessed directly by that program.

The values of local variables are not saved (or restored) by the STORE (or LOAD) monitor command.

Parameters

type	Optional parameter specifying the type of a variable. The acceptable types are:	
	LOC	Location variable (transformation or precision point)
	REAL	Single-precision real variable
	DOUBLE	Double-precision real variable See the description of the GLOBAL program instruction for details on the default type.
variable	Variable name (belt, precision point, real-value, string, or transformation). Each variable can be a simple variable or an array. Array variables must *not* have their indexes specified. If a type is specified, all variables must be of that type.	

Details

This instruction is used to declare variables to be defined only within the current program. That is, a local variable can be referenced only within its own program. Also, the names of local variables can be selected without regard for the names of local variables defined in other programs.

Local variables are allocated only once during program execution, and their values are preserved between successive subroutine calls. These values are also shared if the same program is executed by multiple program tasks.

If a program that uses LOCAL (or global) variables is called by several different program tasks, or called recursively by a single task, the values of those variables can be modified by the different program instances and cause very strange program errors. Therefore, automatic variables should be used for all temporary local variables to minimize the chance of errors. (See the AUTO instruction.)

Variables can be defined as automatic, global, or local. Once a variable has been assigned to a class, an attempt to assign the variable to a different class will result in the error *Attempt to redefine variable class*.

Variables can be defined only once within the same context (automatic, local, or global). Attempting to define a variable more than once (that is, with a different type) will yield the error *Attempt to redefine variable type*. For details, see Data Types and Operators in the *eV+ Language User's Guide*.

Local variables can be referenced with monitor commands such as BPT, DELETE_, DO, HERE, LIST_, POINT, TEACH, TOOL, and WATCH by using the optional context specifier @. The general syntax is:

```
command @task:program command_arguments
```

For more information on specifying program context, see the section Programming eV+ in the *eV+ Language User's Guide*.

Example

Declare the variables **loc.a**, **$ans**, and **i** to be local to the current program:

```
LOCAL loc.a, $ans, i
```

Related Keywords

AUTO program instruction

GLOBAL program instruction

LOCK program instruction

Syntax

LOCK priority

Function

Set the program reaction lock-out priority to the value given.

Usage Considerations

LOCK 0 is assumed whenever program execution is initiated and when a new execution cycle begins.

Changing the priority may affect how reactions are processed. Before using this instruction, be sure you know what reactions are active (and their priorities).

Parameter

priority	Real-valued expression with a value from 0 to 127, which becomes the new reaction lock-out priority.

Details

When a program is EXECUTEd, it is placed on the execution stack. When the program's task becomes the highest priority task in a time slice, the program's priority is set to 0 and it begins execution. During actual execution, a program's task can be suspended at the end of a time slice, in which case the task waits until the next time it is the highest priority task in a time slice. The LOCK instruction does not affect the task priority value within a time slice: It only changes the program priority of an executing program.

Program priority becomes important when a reaction routine (REACT, REACTE, REACTI) is invoked. A program can defer execution of a REACT or REACTI routine by setting the temporary program priority to a value higher than the REACT or REACTI program priority. This is the function of a LOCK instruction. For example, if a LOCK instruction changes the temporary program priority to 20, any REACT or REACTI interrupts with lower priority values are deferred. (REACTE routines cannot be deferred by priority considerations.)

Deferred reactions are not ignored. Every time a new LOCK instruction is processed, any deferred reaction programs are checked to see if their priority is high enough for them to execute. As soon as the program priority is lowered, all pending reaction routines with a higher priority are run according to their relative priority.

The PRIORITY real-valued function can be used to determine the program priority at any time.

> **NOTE:** Although a LOCK instruction can be used to change the program priority within a reaction program, the priority still returns to its prereaction value when a RETURN is executed in the program. This occurs only when executing a RETURN from a reaction program.

Example

Increase the program priority by 10:

```
LOCK PRIORITY+10
```

Related Keywords

PRIORITY real-valued function

REACT program instruction

REACTI program instruction

MAX real-valued function

Syntax

MAX (value, ..., value**)**

Function

Return the maximum value contained in the list of values.

Parameter

value	Each value in the list can be specified as a real-valued constant, variable, or expression.

Details

The list of values provided is scanned for the largest value, and that value is returned by the function.

The sign of each value is considered. Thus, for example, the value -10 is considered larger than -100.

Example

The program instruction:

```
max.value = MAX(x, y, z, 0)
```

sets **max.value** to the largest value of the variables **x**, **y**, and **z**, or to zero if all three variables have values less than zero.

Related Keyword

MIN real-valued function

MC program instruction

Syntax

MC monitor_command

Function

Introduce a monitor command within a command program.

Usage Considerations

The MC instruction can be contained only within a command program. (Command programs can contain *only* MC instructions, blank lines, and comment lines.)

Parameter

monitor_command Any valid eV+ monitor command.

Details

Command programs are created using one of the eV+ editors. To indicate to the editor that a command program, rather than a normal program, is being created, every operation line of a command program must begin with the letters MC (that is, for Monitor Command follows) followed by one or more spaces. As with regular application programs, command programs can contain blank lines and comment lines to add clarity.

Every nonblank line of a command program must contain a monitor command (or a comment). Monitor commands and program instructions cannot be mixed. Program instructions can be included, however, by using the DO command. That is, to include an instruction in a command program, you can type a line with the form **mc do** instruction. See the *eV+ Operating System Reference Guide* for details on monitor commands.

Example

The following command program loads disk files, prepares for execution of a program, and begins the execution. Note that a DO command is used to include a MOVE instruction:

```
1 .PROGRAM setup()
2         MC LOAD C:project
3         MC LOAD B:project.lc
4         MC SPEED 50
5         MC DO MOVE safe.loc
6         MC EXECUTE motion, -1
7 .END
```

Related Keywords

COMMANDS monitor command

MCS program instruction

MCS program instruction

Syntax

MCS string

Function

Invoke a monitor command from an application program.

Parameter

string	String value, variable, or expression that defines one of the eV+ monitor commands listed below.

Details

Normally, monitor commands can be invoked only from the system terminal or from command programs (which contain only monitor commands). The MCS instruction can be used to invoke the following monitor commands from an application program:

DELETE	DELETEL	DELETEM	DELETEP	DELETER
DELETES	FCOPY	LOAD	STORE	STOREL
STOREM	STOREP	STORER	STORES	VRENAME

Using these commands, an application program can store, load, and copy programs to and from disk, and also delete programs from memory to make room for other programs. Similarly, variables can be deleted from memory when they are no longer needed. Also, vision prototypes can be renamed.Loading, storing, and deleting programs and global variables is not interlocked for multi-task access in eV+. Therefore, if you are incorporating multiple MCS instructions in a program, you will need to use TAS interlocks to prevent multiple tasks from issuing the instructions. For details, see the TAS program instruction.

> **NOTE:**If the monitor command specified in the string parameter contains a blank program context (that is, it contains @), any variables listed in the command are treated as though they are referenced within the program containing the MCS instruction. (See the *eV+ Language User's Guide* for more information on program context.)

Program execution is not stopped if an error occurs while processing the monitor command. The ERROR real-valued function can be used after the MCS instruction to check for the occurrence of an error.

> **NOTE:** If a DELETE_ command is used within a subroutine to delete one of the subroutine parameters (that is, one of the variables in the .PROGRAM statement), the variable is not deleted and no error condition is recorded.

Normal output by the monitor command to the system terminal is done if the MCS.MESSAGE system switch is enabled. For example, the LOAD command outputs the .PROGRAM lines from each program loaded. (The MCS.MESSAGE switch is normally disabled.)

If the FCOPY option is used, logical units 5 (disk #1) and 6 (disk #2) must be available. If LOAD or STORE_ is used, logical unit # 5 must be available.

Example

The following program loads a disk file, executes the program in the file, and deletes the program from the system memory. Another program file is then loaded into memory and executed. (Although this simple example can also be implemented with a command program, the following demonstrates use of the MCS instruction in a normal program.)

```
.PROGRAM admin()
        MCS "LOAD C:setup"
        CALL setup
        MCS "DELETEP setup"
        MCS "LOAD C:demo_1"
        CALL demo_main
.END
```

Related Keywords

ERROR real-valued function

MC program instruction

MESSAGES system switch

Syntax

... MESSAGES

Function

Enable or disable output to the system terminal from TYPE instructions.

Details

If this switch is enabled, output from TYPE instructions is displayed on the system terminal. Otherwise, output is suppressed.

By default, this switch is enabled, allowing output to occur.

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

TYPE program instruction

$MID string function

Syntax

$MID (string, first_char**, num_chars)**

Function

Return a substring of the specified string.

Parameters

string	String variable, constant, or expression from which the substring is extracted.
first_char	Optional real-valued expression that specifies the first character of the substring.
num_chars	Real-valued expression that specifies the number of characters to be copied to the substring.

Details

If first_char is omitted or has a value less than or equal to 1, the substring starts with the first character of **string**. If first_char is larger than the length of the input string, the function returns an empty string.

If there are fewer than **num_chars** characters from the specified starting character position to the end of the input string, the output string consists of only the characters up to the end of the input string. That is, no error results and the output string is not extended to the requested length.

Example

The instructions below result in the string variable $substring containing the string cd, since cd is the 2-character string that starts at character position 3 of the string abcde contained in the string variable $string:

```
$string = "abcdef"
$substring = $MID($string, 3, 2)
```

Related Keyword

$UNPACK string function

MIN real-valued function

Syntax

MIN (value, ..., value**)**

Function

Return the minimum value contained in the list of values.

Parameter

value	Each value in the list can be specified as a real-valued constant, variable, or expression.

Details

The list of values provided is scanned for the smallest value, and that value is returned by the function.

The sign of each value is considered. Thus, for example, the value -100 is considered smaller than -10.

Example

The program instruction:

```
min.value = MIN(1000, x, y, z)
```

sets min.value to the smallest value of the variables x, y, and z, or to the value 1000 if all three variables have values greater than 1000.

Related Keyword

MAX real-valued function

MMPS keyword

Syntax

SPEED value MMPS ALWAYS

Function

Specify the units for a SPEED instruction as millimeters per second.

> **NOTE:** To specify units in inches per second, use the IPS conversion factor.

Usage Considerations

MMPS can be used only as a parameter for a SPEED program instruction.

The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.

Speeds specified with the MMPS parameter apply to straight-line motions. Joint-interpolated motions do not maintain the specified tool speed.

Details

This is an optional parameter for the SPEED program instruction, which specifies the units to be used for the speed value. That is, when MMPS is specified in a SPEED instruction, the speed value is interpreted as millimeters/second (for straight-line motions).

See the description of the SPEED program instruction for further details on setting motion speeds with the IPS conversion factor.

Example

Set the default program speed for straight-line motions to 10 millimeters per second (assuming the monitor speed is set to 100):

```
SPEED 10 MMPS ALWAYS
```

Related Keywords

IPS keyword

SPEED program instruction

MOD operator

Syntax

... value MOD value ...

Function

Compute the modulus of two values.

Details

The MOD operator operates on two values, resulting in a value that is the *remainder* after dividing the first value by the second value. (The second value cannot be zero.)

For details on how operators are evaluated within expressions, see the Order of Evaluation.

Examples

Return 1 (5/2 is 2 with a remainder of 1):

```
5 MOD 2
```

Return 0 (81/27 is 3 with a remainder of 0):

```
81 MOD 27
```

MOVE and MOVES program instruction

Syntax

MOVE location

MOVES location

Function

Initiate a robot motion to the position and orientation described by the given location.

Usage Considerations

MOVE causes a joint-interpolated motion.

MOVES causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Parameter

location	Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

Details

The MOVE instruction causes a joint-interpolated motion. That is, intermediate set points between the initial and final robot locations are computed by interpolating between the initial and final joint positions. Any changes in configuration requested by the program (for example, by a LEFTY instruction) are executed during the motion.

The MOVES instruction causes a straight-line motion. During such a motion the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are allowed during straight-line motions.

Examples

MOVE #pick	Move by joint-interpolated motion to the location described by the precision point #pick.

MOVES ref:place	Move along a straight-line path to the location described by the compound transformation ref:place.

Related Keywords

APPRO program instruction

APPROS program instruction

DEPART program instruction

DEPARTS program instruction

MOVEC program instruction

SELECT program instruction

SELECT real-valued function

MOVEC program instruction

Syntax

MOVEC(angle, turn) **location1, location2**

MOVEC(angle, turn**) center**

Function

Initiate a circular/arc-path robot motion using the positions and orientations described by the given locations.

Usage Considerations

This instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing this instruction causes an error.

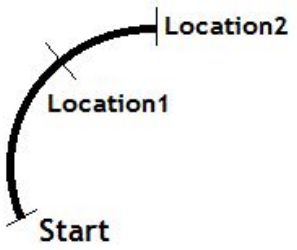
Parameters

angle	Real-valued expression that specifies the angle of the arc in degrees. This parameter is optional if *location2* is specified. *angle* can be a positive or negative number but must be within [-360, +360].
turn	Optional boolean expression that specifies whether the tool should rotate with the arc. If *turn* is omitted or zero, then the tool orientation will stay constant with a MOVEC(angle) center syntax and end at the orientation of *location2* for a MOVEC location1, location2 syntax. If *turn* is non-zero, the tool orientation will be rotated by the angle of the arc around the axis of the circle and maintain a constant orientation relative to the trajectory. This is useful for dispensing applications.
center	Transformation, precision point, location function, or compound transformation that specifies the center of the circle.
location1	Transformation, precision point, location function, or compound transformation that specifies an intermediate location on the circle/arc through which the robot is to move.
location2	Optional transformation, precision point, location function, or compound transformation that specifies the end-point of the circle/arc to which the robot is to move. If this parameter is not supplied, then *angle* must be specified.

Details

MOVEC*(angle,turn)* ***location1, location2***

This MOVEC program instruction syntax is designed to create a circle/arc path that starts from the current robot position or, in case of Continuous Path, the current robot destination, and ends at a location defined by *location2*. The intermediate location (*location1*) is used to define the plane of the circle and the angle of the arc. See the following figure.



MOVEC with Location1 and Location2

If the three points are aligned or two of them coincide, MOVEC will cause a straight-line motion instead of creating a circle or arc.

With this syntax, the orientation of *location1* is not used.

If *angle* is specified then the robot will move by *angle* degrees and not necessarily end up at *location2*. In other words, the *angle* has higher priority than *location2* in defining the final position.

When *angle* is specified, the orientation of *location2* is ignored. The final orientation is determined entirely by the *turn* parameter: if it is omitted or zero, then the final orientation will be the orientation of the start position; if *turn* is non-zero, then the final orientation is the one of the start position rotated by *angle* around the axis of the arc.

When *turn* is non-zero, MOVEC will generate an *invalid orientation* error for 4-axis robots (like the Adept Cobra and Quattro robots, or Adept Python linear modules) if the plane of the circle is not parallel to the XY plane of the Tool Center Point.

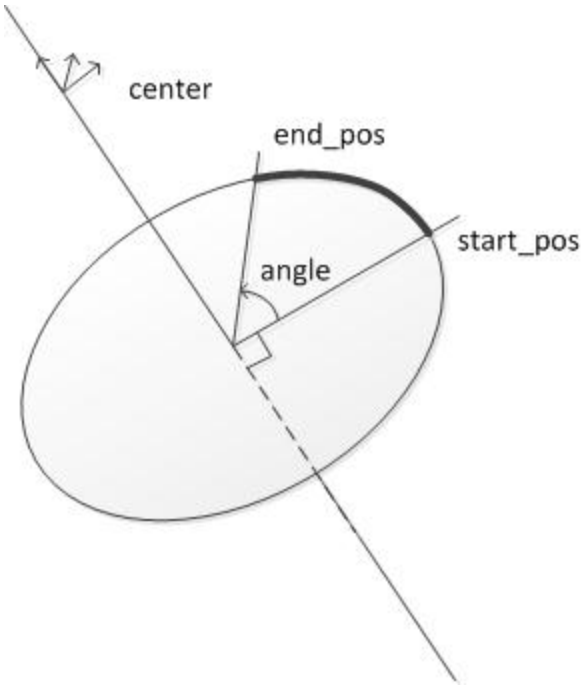
As with straight-line motion, circular motion is compatible with multi-turn rotation. This means that if location2 is a precision point, the multi-turn joint can rotate more than 360 degrees.

MOVEC(angle*,turn****) center***

This MOVEC program instruction syntax is designed to create a circle/arc path that starts from the current robot position or, in case of Continuous Path, the current robot destination. The path is centered around the *center* location; the end location is specified with *angle* degrees.

The plane of the circle is defined as the plane passing through the start position and parallel to the XY plane of the *center* location. In other words, if the Z-orientation of the *center* location is not perpendicular to the straight line passing through *center* and the start position, then the center of the circle is not the location *center*; rather, it is the intersection of the Z-axis of *center* and a plane that is perpendicular to this axis and passes through the start position.

After the actual center of the circle is defined, the radius of the circle is simply the distance from the start position to the actual center.



MOVEC with Angle and Center

Examples

The following example shows MOVEC being used with transformations.

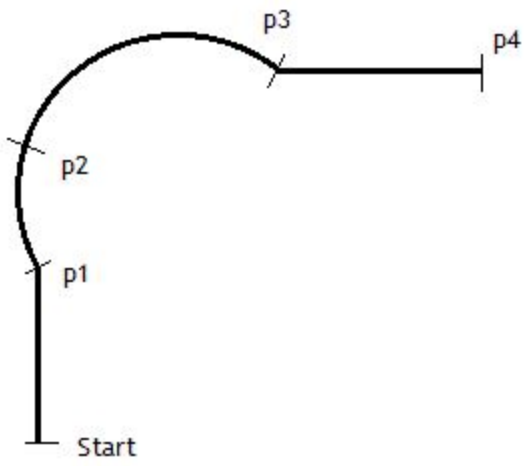
```
MOVEC loc1, loc2
```

Continuous Path Example

The following example shows MOVEC being used with Continuous Path. Also, see the following figure.

```
MOVES p1
```

```
MOVEC p2, p3
MOVES p4
BREAK
```

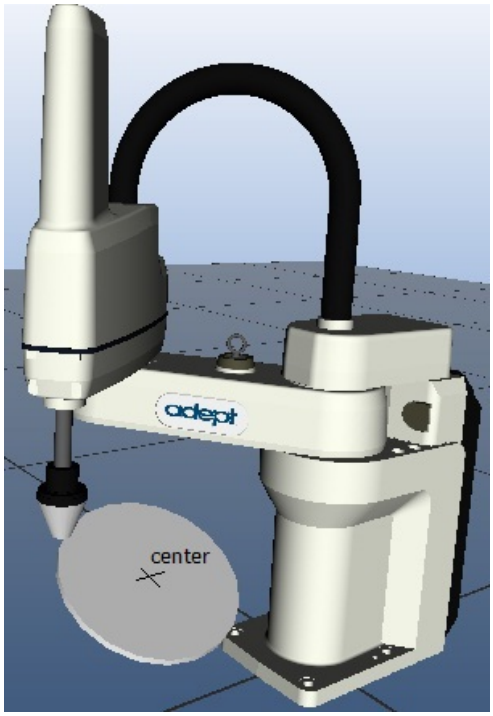


MOVEC Combined with Continuous Path

Full Circle Example

The following example shows MOVEC being used to create a full circle with an Adept Cobra s600 robot.

```
SET center = TRANS(300,0,210,90,30,0)
SET start_pos = TRANS(420, 0, 210, 0, 180, 0)

MOVES start_pos
BREAK

; Do a full circle
MOVEC(360) center
BREAK
```



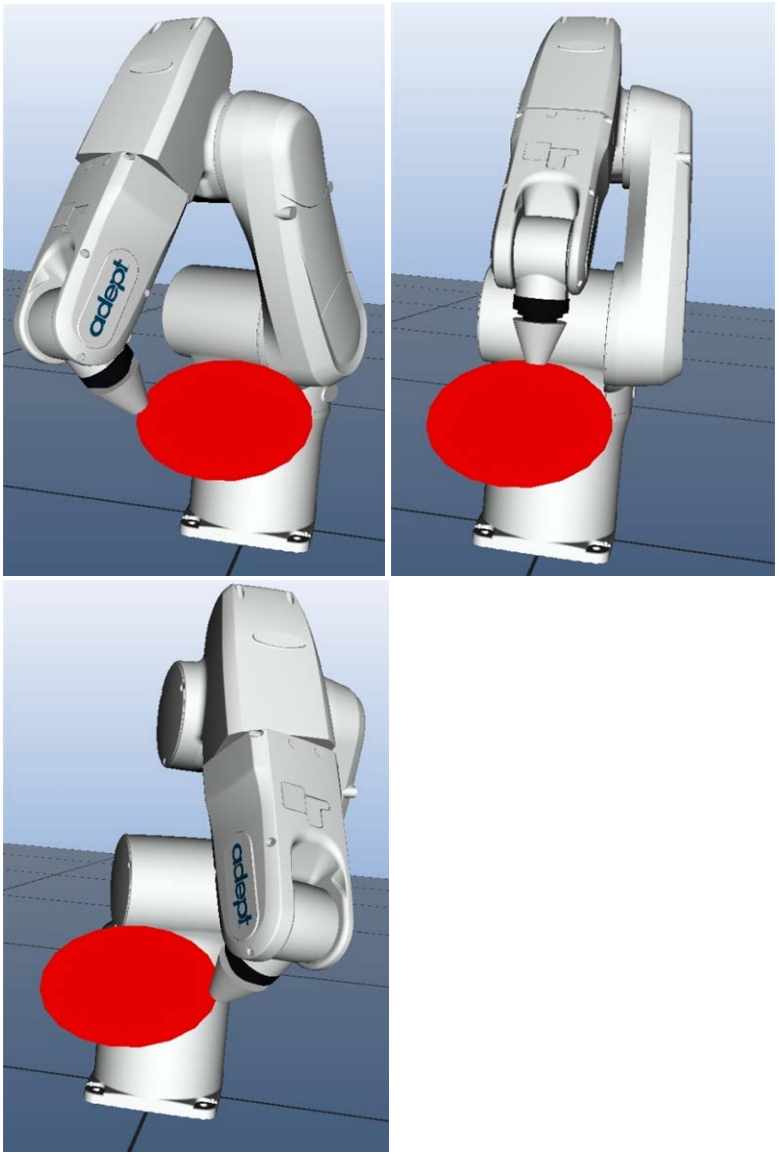
MOVEC Creating a Full Circle.

NOTE: In the previous figure, note that "MOVEC(360,1) center" would have returned an *invalid orientation* error because the Adept Cobra robot cannot maintain a constant orientation relative to a non-horizontal circle.

Half Circle Examples

The following example shows MOVEC being used to create a half circle with rotating orientation for dispensing with an Adept Viper s650.

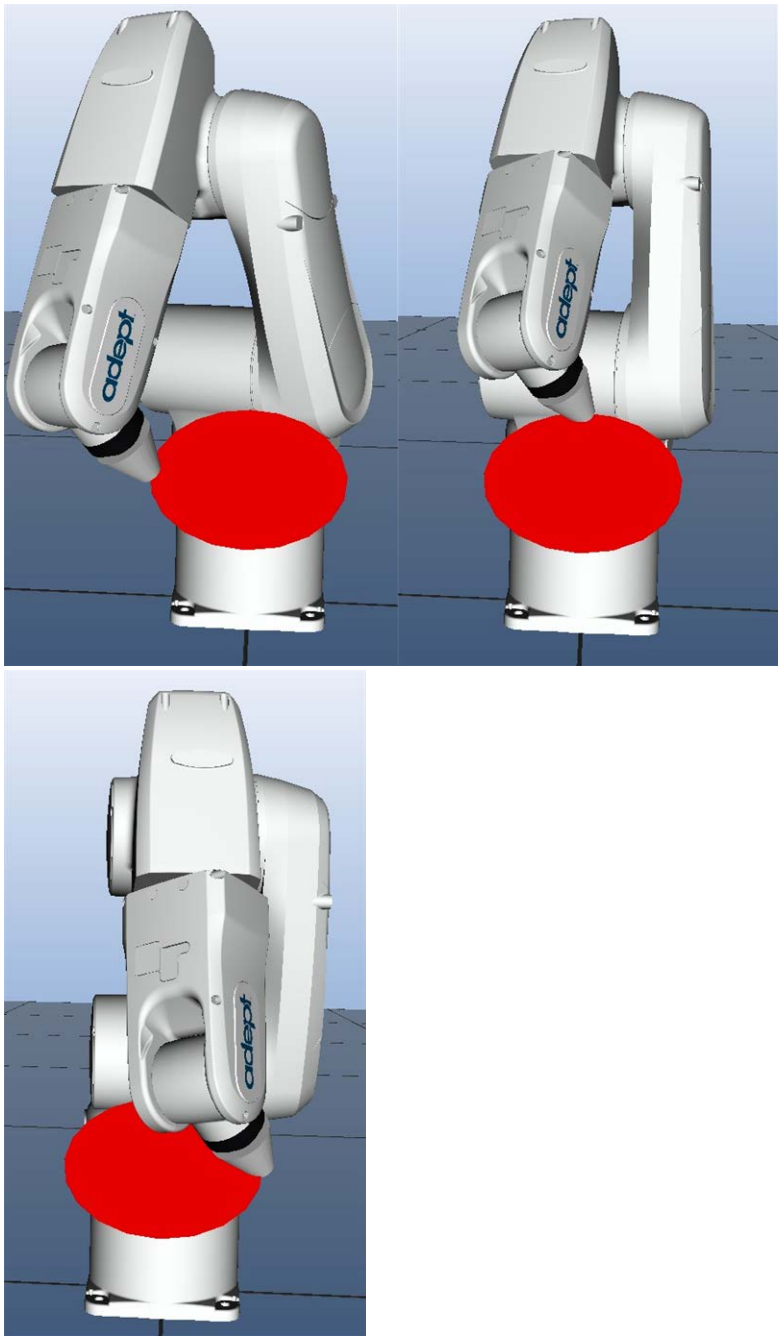
```
SET center = TRANS(300,0,250,0,-150,0)
SET start_pos = center:TRANS(0,0,0,-90):TRANS(100,,,,-30)

MOVES start_pos
BREAK

; Do a half circle with rotating orientation
MOVEC(180,TRUE) center
BREAK
```



MOVEC Creating a Half Circle with rotating orientation (e.g. the "turn" parameter is set).

By changing the last 3 lines of the code, the example can be modified to create the same motion but without rotating the orientation, as follows:

```
; Same motion without rotating the orientation
MOVEC(180) center
BREAK
```



MOVEC Creating a Half Circle with constant orientation (e.g. the "turn" parameter is omitted).

Related Keywords

APPRO program instruction

APPROS program instruction

DEPART program instruction

DEPARTS program instruction

MOVE program instruction

MOVES program instruction

SELECT program instruction

SELECT real-valued function

MOVE and MOVES program instruction

Syntax

MOVE location

MOVES location

Function

Initiate a robot motion to the position and orientation described by the given location.

Usage Considerations

MOVE causes a joint-interpolated motion.

MOVES causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task as long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Parameter

location	Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

Details

The MOVE instruction causes a joint-interpolated motion. That is, intermediate set points between the initial and final robot locations are computed by interpolating between the initial and final joint positions. Any changes in configuration requested by the program (for example, by a LEFTY instruction) are executed during the motion.

The MOVES instruction causes a straight-line motion. During such a motion the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are allowed during straight-line motions.

Examples

MOVE #pick	Move by joint-interpolated motion to the location described by the precision point #pick.

MOVES ref:place	Move along a straight-line path to the location described by the compound transformation ref:place.

Related Keywords

APPRO program instruction

APPROS program instruction

DEPART program instruction

DEPARTS program instruction

MOVEC program instruction

SELECT program instruction

SELECT real-valued function

MULTIPLE program instruction

Syntax

MULTIPLE *ALWAYS*

Function

Allow full rotations of the robot wrist joints.

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.

This is the default state of the eV+ system. MULTIPLE ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The MULTIPLE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the MULTIPLE instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes MULTIPLE as the default condition. That is, if ALWAYS is specified, MULTIPLE will remain in effect continuously until disabled by a SINGLE instruction. If ALWAYS is not specified, the MULTIPLE instruction applies only to the next robot motion.

Details

While MULTIPLE is in effect, full rotations of the wrist joints are used, as required, during motion planning and execution.

The MULTIPLE setting is ignored if NOOVERLAP is in effect.

Related Keywords

CONFIG real-valued function

NOOVERLAP program instruction

OVERLAP program instruction

SELECT program instruction

SELECT real-valued function

SINGLE program instruction

NETWORK real-valued function

Syntax

NETWORK (component, code)

Function

Return network status and IP address information

Parameters

component Real-valued expression that identifies the component of the network that is of interest.

1 = TCP
3 = FTP

code Real-valued expression that further identifies the information desired. This value is only used for TCP:

0 = Return status value as described later (default).
1 = Return AD1*256 + AD2
2 = Return AD3*256 + AD4
3 = Return NM1*256 + NM2
4 = Return NM3*256 + NM4

11 = Return AD1
12 = Return AD2
13 = Return AD3
14 = Return AD4
15 = Return NM1
16 = Return NM2
17 = Return NM3
18 = Return NM4

where AD*n* is the nth byte of the IP address and NM*n* is the nth byte of the Network Mask

Details

This function returns one of the following values if status is requested (that is, if the code argument is omitted or set to 0):

Value	Meaning
0	Hardware not present, or license not installed
-1	Hardware present and license detected
1	Driver is running

NEXT program instruction

Syntax

NEXT count

Function

Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.

Usage Considerations

This instruction can be used with the FOR, WHILE, and DO control structures.

Parameter

count	Optional integer specifying the number of nested structures to branch to the END of (expressions and variables are not acceptable).

Details

When a NEXT instruction is processed with count = 1, execution continues at the END of the control structure. If count > 1, execution continues at the END of count number of nested control structures.

Example

If error = 1, branch to the END of the innermost control structure. If error = 2, branch to the END of the outermost control structure:

```
45  FOR i = 1 to 20
46      FOR j = 1 to 10
47          FOR k = 10 to 50
48              IF error == 1 THEN
49                  NEXT              ;branch to step 54
50              END
51              IF error == 2 THEN
52                  NEXT 3            ;branch to step 56
53              END
54          END
55      END
56  END
57
```

Related Keywords

DO program instruction

EXIT program instruction

FOR program instruction

WHILE program instruction

NOFLIP program instruction

Syntax

NOFLIP

Function

Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a positive value.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a no-flip configuration, this instruction is ignored by the robot.

The NOFLIP instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the NOFLIP instruction causes an error.

For more details, see the description of the FLIP program instruction.

Related Keywords

CONFIG real-valued function

FLIP program instruction

SELECT program instruction

SELECT real-valued function

NONULL program instruction

Syntax

NONULL *ALWAYS*

Function

Instruct the eV+ system not to wait for position errors to be nulled at the end of continuous-path motions.

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.

NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The NONULL instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the NONULL instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes NONULL as the default condition. That is, when ALWAYS is included in a NONULL instruction, NONULL remains in effect continuously until disabled by a NULL instruction. If ALWAYS is not specified, the NONULL instruction applies only to the next robot motion.

Details

When NONULL is in effect and a BREAK in the robot motion occurs, eV+ does not wait for the servos to signal that all moving joints have reached their specified positions before it begins the next motion. That is, at the end of the allotted time, eV+ assumes that the joints have all reached their final positions and starts commanding the next motion.

Like COARSE mode, this mode allows faster motion if high final-position accuracy is not required. However, since no position-error checking is done, large position errors can occur.

Related Keywords

COARSE program instruction

CONFIG real-valued function

DELAY.IN.TOL program instruction

FINE program instruction

NULL program instruction

SELECT program instruction

SELECT real-valued function

NOOVERLAP program instruction

Syntax

NOOVERLAP *ALWAYS*

Function

Generate a program error if a subsequent motion is planned that causes a selected multi-turn axis to move more than ±180 degrees to avoid a limit stop.

Usage Considerations

NOOVERLAP applies to the operation of the following robots/joints:

- For Adept Viper and PUMA robots: joints 1, 4, and the final joint
- For all SCARA robots: joint 4
- For Adept Quattro robots: tool rotation

OVERLAP ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The NOOVERLAP instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the NOOVERLAP instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes NOOVERLAP as the default condition. That is, if ALWAYS is specified, NOOVERLAP remains in effect continuously until disabled by an OVERLAP instruction. If ALWAYS is not specified, the NOOVERLAP instruction applies only to the next robot motion.

Details

When NOOVERLAP is set, and the transformation destination of a joint-interpolated or straight-line motion requires that a multiple-turn axis rotate more than ±180 degrees to avoid a limit stop, a program error will occur (and the motion will not be performed). If the destination is specified as a precision point, this test is not performed.

In general, given a transformation destination, a multiple-turn axis normally attempts to move to a new position by moving in the direction that requires less than 180 degrees of

motion. The only conditions that force an axis to make a larger change are if SINGLE is specified, or if a software limit stop would be violated.

When NOOVERLAP is set, the setting of SINGLE or MULTIPLE mode is ignored.

As with other user program errors, the error condition generated as a result of the NOOVERLAP test can be trapped by a standard REACTE subroutine if desired.

Related Keywords

MULTIPLE program instruction

OVERLAP program instruction

SELECT program instruction

SELECT real-valued function

SINGLE program instruction

NORMAL transformation function

Syntax

NORMAL (transformation_value)

Function

Correct a transformation for any mathematical round-off errors.

Usage Considerations

For most robot programs, transformation normalizing never has to be performed.

Parameter

transformation_value	Transformation, transformation valued function, or compound transformation whose value is to be normalized.

Details

Use this function after a lengthy series of computations that modifies a transformation value. For instance, a procedural motion that incrementally changes the orientation of a transformation should occasionally normalize the resultant value. Within a transformation, the orientation of the robot is represented by three perpendicular unit vectors. Because of the small inaccuracies that occur in computer computations, after being incrementally modified many times, these vectors can become nonperpendicular or not of unit length.

The NORMAL function returns a transformation value that is essentially the same as the input argument but has the orientation portion of the value corrected for any small buildup of computational errors that may have occurred.

NOT operator

Syntax

... NOT value ...

Function

Perform logical negation of a value.

Usage Considerations

The word "not" cannot be used as a program name or variable name.

Details

The NOT operator operates on a single value, converting it from logically true to false, and vice versa. If the single value is zero, a -1.0 (TRUE) is returned. Otherwise, a 0.0 (FALSE) value is returned.

Refer to the *eV+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

```
IF NOT initialized THEN        ;If the variable "initialized" has
a
    CALL appl.setup()          ;FALSE value, the instructions in
the
    initialized = TRUE         ;IF structure will be executed.
END
```

NOT.CALIBRATED system parameter

Syntax

... NOT.CALIBRATED

Function

Indicate (or assert) the calibration status of the robots connected to the system.

Usage Considerations

The current value of the NOT.CALIBRATED parameter can be determined with the PARAMETER monitor command or real-valued function.

You can modify the value of this parameter at any time; refer to the next section for details.

The parameter name can be abbreviated.

Details

The value of this parameter, which can range from -1 to 32767, should be interpreted as a bit mask. Bits 1 through 15 correspond to robots 1 through 15, respectively. For example, the following values have the following interpretations:

Value of parameter	Interpretation
0	All robots are calibrated.
1	Robot 1 is not calibrated.
3	Robots 1 and 2 are not calibrated.
7	Robots 1 through 3 are not calibrated.

On power-up, this parameter is set to indicate that all installed robots are not calibrated. If a robot is not connected or not defined, its NOT.CALIBRATED bit is always off.

The CALIBRATE command and instruction attempt to calibrate any enabled ROBOT that has its NOT.CALIBRATED bit set.

When the calibration operation completes, the NOT.CALIBRATEd bits are updated as appropriate. For example, consider a system that has only one robot installed. If the CALIBRATE command is issued, and it succeeds, NOT.CALIBRATED is set to 0. If three robots are connected, and the CALIBRATE command succeeds in calibrating robots 1 and 2, but not robot 3, NOT.CALIBRATED is set to 4 (binary 100-robots 1 and 2 calibrated, 3 not calibrated).

The purpose of this parameter is to allow one of the bits to be *set* to force the corresponding robot to be calibrated the next time a CALIBRATE command or instruction is executed. This parameter can also be used to determine the calibration status of the robot(s).

The parameter value can be changed at any time. The following rules describe how a new asserted value is treated:

- If the new value asserts that a robot is not calibrated, the eV+ system behaves as if the robot is not calibrated whether or not the servo software believes that the robot is not calibrated.
- If the new value asserts that a robot is calibrated, the servo software is checked and eV+ tracks the calibrated/not calibrated state indicated by the servo software for that robot.

NOTE: It is usually not meaningful to use PARAMETER NOT.CALIBRATED to clear a bit.

Examples

Mark all installed robots as uncalibrated:

```
PARAMETER NOT.CALIBRATED = -1
```

Mark only robots 1 and 2 as uncalibrated:

```
PARAMETER NOT.CALIBRATED = 3
```

Related Keywords

CALIBRATE monitor command

CALIBRATE program instruction

PARAMETER monitor command

PARAMETER program instruction

PARAMETER real-valued function

ROBOT system switch

NULL program instruction

Syntax

NULL *ALWAYS*

Function

Instruct the eV+ system to wait for position errors to be nulled at the end of continuous path motions.

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.

This is the default state of the eV+ system. NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The NULL instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the NULL instruction causes an error.

The word "null" cannot be used as a program name or variable name.

Parameter

ALWAYS	Optional qualifier that establishes NULL as the default condition. That is, if ALWAYS is specified, NULL remains in effect continuously until disabled by a NONULL instruction. If ALWAYS is not specified, the NULL instruction applies only to the next robot motion.

Details

When NULL is in effect and a BREAK in the robot motion occurs, eV+ waits for the servos to signal that all moving joints have reached their specified positions before it begins the next motion. The accuracy to which the electronics verify that all joints have reached their destination positions is determined by the COARSE and FINE program instructions.

Related Keywords

COARSE program instruction

DELAY.IN.TOL program instruction

FINE program instruction

NONULL program instruction

NULL program instruction

SELECT program instruction

SELECT real-valued function

NULL transformation function

Syntax

NULL

Function

Return a null transformation value-one with all zero components.

Usage Considerations

The word "null" cannot be used as a program name or variable name.

Details

A null transformation corresponds to a null vector (X = Y = Z = 0) and no rotation (yaw = pitch = roll = 0). Such a transformation is useful, for example, with a SHIFT function to create a transformation representing a translation with no rotation.

Example

Define a new transformation (new.loc) to be the result of shifting an existing transformation (old.loc) in the World coordinate directions.

```
new.loc = SHIFT(NULL BY x.shift,y.shift,z.shift):old.loc
```

Determine the length of the vector described by the transformation test.loc.

```
dist = DISTANCE(NULL, test.loc)
```

Related Keywords

CONFIG real-valued function

NULL program instruction

OFF real-valued function

Syntax

OFF

Function

Return the value used by eV+ to represent a logical false result.

Usage Considerations

The word "off" cannot be used as a program name or variable name.

Details

This named constant is useful for situations where on and off conditions need to be specified. The value returned is 0.

This function is equivalent to the FALSE function.

Related Keywords

FALSE real-valued function

ON real-valued function

ON real-valued function

Syntax

ON

Function

Return the value used by eV+ to represent a logical true result.

Usage Considerations

The word "on" cannot be used as a program name or variable name.

Details

This named constant is useful for situations where on and off conditions need to be specified. The value returned is -1.

This function is equivalent to the TRUE function.

Related Keywords

OFF real-valued function

TRUE real-valued function

OPEN program instruction

Syntax

OPEN

OPENI

Function

Open the robot gripper.

Usage Considerations

OPEN causes the hand to open *during* the next robot motion.

OPENI causes a BREAK in the current continuous-path motion and causes the hand to open *immediately* after the current motion completes.

The OPEN instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The OPENI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions cause the control valves for the pneumatic hand to receive a signal to open. If the OPEN instruction is used, the signal is sent when the next robot motion begins.

> **NOTE:** You can use the Robot Configuration Utility program to set the digital signals that control the pneumatic hand. The utility program is on the Adept Utility Disk. See the manual *Instructions for Adept Utility Programs* for information on use of the program.

The OPENI instruction differs from OPEN in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Examples

During the next robot motion, cause the pneumatic control valves to assume the open state:

```
OPEN
```

Cause the pneumatic control valves to assume the open state at the conclusion of the current motion:

```
OPENI
```

Related Keywords

CLOSE program instruction

CLOSEI program instruction

HAND.TIME system parameter

RELAX program instruction

RELAXI program instruction

SELECT program instruction

SELECT real-valued function

OPEN program instruction

Syntax

OPEN

OPENI

Function

Open the robot gripper.

Usage Considerations

OPEN causes the hand to open *during* the next robot motion.

OPENI causes a BREAK in the current continuous-path motion and causes the hand to open *immediately* after the current motion completes.

The OPEN instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The OPENI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions cause the control valves for the pneumatic hand to receive a signal to open. If the OPEN instruction is used, the signal is sent when the next robot motion begins.

> **NOTE:** You can use the Robot Configuration Utility program to set the digital signals that control the pneumatic hand. The utility program is on the Adept Utility Disk. See the manual *Instructions for Adept Utility Programs* for information on use of the program.

The OPENI instruction differs from OPEN in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Examples

During the next robot motion, cause the pneumatic control valves to assume the open state:

```
OPEN
```

Cause the pneumatic control valves to assume the open state at the conclusion of the current motion:

```
OPENI
```

Related Keywords

CLOSE program instruction

CLOSEI program instruction

HAND.TIME system parameter

RELAX program instruction

RELAXI program instruction

SELECT program instruction

SELECT real-valued function

OR operator

Syntax

... value OR value ...

Function

Perform the *logical* OR operation on two values.

Details

The OR operator operates on two values, resulting in their logical OR combination. For example, during the OR operation

```
c = a OR b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	->	FALSE
FALSE	TRUE	->	TRUE
TRUE	FALSE	->	TRUE
TRUE	TRUE	->	TRUE

That is, the result is TRUE if either (or both) of the two operand values is logically TRUE. To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Example

In the following sequence, the instructions immediately following the IF instruction are executed if either ready is TRUE (that is, nonzero) or count equals 1. The instructions are not executed if both ready is FALSE and count is not equal to 1.

```
IF ready OR (count == 1) THEN
  .
  .
  .
END
```

Related Keywords

AND operator

BOR operator

XOR operator

OUTSIDE real-valued function

Syntax

OUTSIDE (low, test, high)

Function

Test a value to see if it is outside a specified range.

Parameters

low	Real value, expression, or variable specifying the lower limit of the range to be tested.
test	Real value, expression, or variable to test against the range.
high	Real value, expression, or variable specifying the upper limit of the range to be tested.

Details

Returns TRUE (-1) if **test** is less than **low** or greater than **high**. Returns FALSE (0) otherwise.

Related Keywords

MAX real-valued function

MIN real-valued function

OVERLAP program instruction

Syntax

OVERLAP *ALWAYS*

Function

Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.

Usage Considerations

OVERLAP applies to the operation of the following robots/joints:

- For Adept Viper and PUMA robots: joints 1, 4, and the final joint
- For all SCARA robots: joint 4
- For Adept Quattro robots: tool rotation

This is the default state of the eV+ system. OVERLAP ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The OVERLAP instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the OVERLAP instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes OVERLAP as the default condition. That is, if ALWAYS is specified, OVERLAP remains in effect continuously until disabled by a NOOVERLAP instruction. If ALWAYS is not specified, the OVERLAP instruction applies only to the next robot motion.

Details

If OVERLAP is specified, the settings of SINGLE and MULTIPLE affect the robot motion.

When OVERLAP is set, and the transformation destination of a joint-interpolated or straight-line motion requires that a multiple-turn axis rotate more than ±180 degrees, the motion is executed without generating a program error.

OVERLAP disables the limit-error checking of NOOVERLAP. The OVERLAP setting is applied whenever program execution is initiated and when a new execution cycle begins.

Related Keywords

MULTIPLE program instruction

NOOVERLAP program instruction

SELECT program instruction

SELECT real-valued function

SINGLE program instruction

PACK program instruction

Syntax

PACK string_array[index**], first_char, num_chars = string**

PACK string_var, first_char, num_chars = string

Function

Replace a substring within an array of (128-character) string variables, or within a (nonarray) string variable.

Parameters

string_array	String array variable that is modified by the substring on the right-hand side of the equal sign. Each element within the string array is assumed to be 128 characters long (see below).
index	Optional integer value that identifies the first array element to be considered. The **first_char** value is interpreted relative to the element specified by this index. If no index is specified, element zero is assumed.
string_var	String variable that is modified by the substring on the right-hand side of the equal sign.
first_char	Real-valued expression that specifies the position of the first character of the substring within the string array. A value of 1 corresponds to the first character of the specified string array element. This value must be greater than zero. The value of **first_char** can be greater than 128. In that case the array element accessed follows the element specified in the function call. For example, a value of 130 corresponds to the second character in the array element following that specified by index.
num_chars	Real-valued expression that specifies the number of characters to be copied from the string to the array. This value can range from 0 to 128.
string	String variable, constant, or expression from which the substring is to be extracted. The string must be at least **num_chars** long.

Details

This instruction replaces a substring within an array of strings or within a string variable. When an array of strings is being modified, the substring is permitted to overlap two elements of the string array. For example, a 10-character substring whose first character is to replace the 127th character in element [3] supersedes the last two characters in element [3] and the first eight characters of element [4].

If the array element to be modified is not defined, the element is created and filled with ASCII NUL characters (^H00) up to the specified start of the substring. Similarly, if the array element to be modified is too short, the string is padded with ASCII NUL characters to the start of the substring.

In order to efficiently access the string array, this function assumes that all of the array elements, from the start of the array until the element before the element accessed, are defined and are 128 characters long. For multidimensional arrays, only the right-most array index is incremented to locate the substring. Thus, for example, element [2,3] is followed by element [2,4].

When a string variable is modified, the replacement is done in a manner similar to that for an individual array element. However, an error results if the operation causes the string to be longer than 128 characters.

Example

The instruction below replaces 11 characters within the string array $list[ ]. The replacement is specified as starting in array element $list[3]. However, since the first character replaced is to be number 130, the 11-character substring actually replaces the second through 12th characters of $list[4].

```
PACK $list[3], 130, 11 = $string
```

Related Keywords

$MID string function

$UNPACK string function

PANIC program instruction

Syntax

PANIC

Function

Simulate an external E-stop or panic button press; stop all robots immediately, but do not turn off HIGH POWER.

Usage Considerations

If the eV+ system is controlling more than one robot, all the robots are stopped.

This instruction has no effect on nonrobot systems.

Details

This instruction performs the following actions:

- Immediately stops robot motion
- Stops execution of the robot control program if the robot is attached and no REACTE has been executed to enable program processing of error.
- Causes *PANIC command* to appear on the monitor screen

Unlike pressing the emergency stop button on the pendant, high power is left turned on after a PANIC instruction is processed.

Related Keywords

ABORT monitor command

ABORT program instruction

ESTOP program instruction

ESTOP monitor command

PANIC monitor command

PARAMETER program instruction

Syntax

PARAMETER parameter_name = value

PARAMETER parameter_name[index**] = value**

Function

Set the value of a system parameter.

Usage Considerations

If the specified parameter accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the parameter array are assigned the value given.

Parameters

parameter_name	Name of the parameter whose value is to be modified.
index	For parameters that can be qualified by an index, this is an optional real value, variable, or expression that specifies the specific parameter element of interest (see above).
value	Real value, variable, or expression defining the value to be assigned to the system parameter.

Details

This instruction sets the given system parameter to the value on the right. The parameter name can be abbreviated to the minimum length that identifies it uniquely.

> **NOTE:** A regular assignment statement cannot be used to set the value of a system parameter.

The parameter names acceptable with the standard eV+ system are summarized in the section Parameters in the *eV+ Language User's Guide*.

Other system parameters are available when options are installed. Refer to the option documentation for details. For example, the parameters associated with the AdeptVision options are described in the section Descriptions of Vision Keywords in the *AdeptVision Reference Guide*.

Example

Set the TERMINAL system parameter to 4:

```
PARAMETER TERMINAL = 4
```

Related Keywords

BELT.MODE system parameter

HAND.TIME system parameter

NOT.CALIBRATED system parameter

PARAMETER monitor command

PARAMETER real-valued function

Syntax

PARAMETER (parameter_name)

PARAMETER (parameter_name[index])

Function

Return the current setting of the named system parameter.

Parameters

parameter_name	Name of the system parameter whose value is to be returned.
index	For parameters that can be qualified by an index, this is a (required) real value, variable, or expression that specifies the specific parameter element of interest.

Details

This function returns the current setting of the given system parameter. The parameter name can be abbreviated to the minimum length that identifies it uniquely.

Other system parameters are available when options are installed. Refer to the option documentation for details. For example, the parameters associated with the AdeptVision options are described in the section Descriptions of Vision Keywords in the *AdeptVision Reference Guide*.

Examples

The following example illustrates how the current setting of the TERMINAL parameter can be displayed on the system terminal during program execution:

```
TYPE "The TERMINAL parameter is set to", PARAMETER(TERMINAL)
```

The PARAMETER function can also be used in any expression to include the value of a parameter. For example, the following program statement can be used to increase the time delay for hand actuation:

```
PARAMETER HAND.TIME = PARAMETER(HAND.TIME) + 0.15
```

Note that the left-hand occurrence of PARAMETER is the instruction name and the right-hand occurrence is the function name.

Related Keywords

BELT.MODE system parameter

HAND.TIME system parameter

NOT.CALIBRATED system parameter

PARAMETER monitor command

PARAMETER program instruction

PAUSE program instruction

Syntax

PAUSE

Function

Stop program execution but allow the program to be resumed.

Usage Considerations

Unlike HALT and STOP, the PAUSE instruction does not force FCLOSE or DETACH on the disk or serial communication logical units. If the program has a file open and you decide not to continue execution of the current program, you should issue a KILL command (with the appropriate task number) to close all files and detach all logical units.

Details

Causes a BREAK and terminates execution of the application program, displaying the message (PAUSED). Execution can subsequently be continued by typing **proceed** and the appropriate task number, and pressing the RETURN key.

When debugging a program, a PAUSE instruction can be inserted to stop program execution temporarily while the values of variables are checked.

> **NOTE:** Any robot motion in progress when a PAUSE instruction is processed completes normally.

Related Keywords

HALT program instruction

KILL monitor command

KILL program instruction

PROCEED monitor command

STOP program instruction

#PDEST precision-point function

Syntax

#PDEST

Function

Return a precision-point value representing the planned destination location for the current robot motion.

Usage Considerations

The #PDEST function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the #PDEST function does not generate an error because of the absence of a robot. However, the information returned by the function may not be meaningful.

The name "pdest" cannot be used for a variable or a program.

Details

The #PDEST function can be used to determine the robot's destination before its motion was interrupted.

The #PDEST function is equivalent to the DEST transformation function and can be used interchangeably with DEST, depending upon the type of location information that is desired. Please refer to the description of the DEST function for more information on the use of both the #PDEST and DEST functions.

Related Keywords

DEST transformation function

HERE transformation function

SELECT program instruction

SELECT real-valued function

PDNT.CLEAR program instruction

Syntax

PDNT.CLEAR

Function

Clears the current notification window or custom message window on the T20 pendant, if any, and returns the T20 pendant back to the Home screen.

Usage Considerations

Parameters

None

Details

None

Example

The following code:

```
PDNT.CLEAR
```

Clears the screen on the T20 pendant.

Related Keywords

PDNT.WRITE program instruction

PDNT.NOTIFY program instruction

PDNT.NOTIFY program instruction

Syntax

PDNT.NOTIFY $title, $msg

Function

Creates a pendant notification.

Usage Considerations

The pendant does not need to be attached using an ATTACH instruction prior to using this function.

Parameters

$title	Optional string constant, variable, or expression that contains the title of the pendant notification.
$msg	Optional string constant, variable, or expression that contains the message of pendant notification.

Details

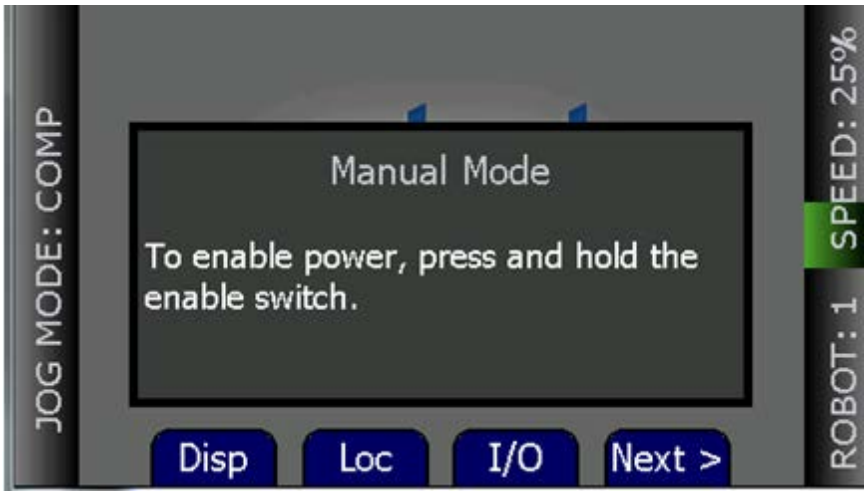
PDNT.NOTIFY is used to create a simple notification box on the T20 Pendant screen that can be cleared by pressing the OK or Cancel buttons on the pendant or with a eV+call to PDNT.CLEAR.

Example

The following code:

```
PDNT.NOTIFY "Manual Mode", "To enable power, press and hold the
enable switch."
```

Creates the screen:



Related Keywords

PDNT.WRITE program instruction

PDNT.CLEAR program instruction

PDNT.WRITE program instruction

Syntax

PDNT.WRITE (msgsize) $title, $msg, $f1, $f2, $f3, $f4

Function

Sets the pendant's Custom Message screen.

Usage Considerations

The pendant does not need to be attached using an ATTACH instruction prior to using this function.

Parameters

msgsize	Optional Real value, variable, or expression whose value represents the array size of $msg.
$title	Optional string constant, variable, or expression that contains the title of the pendant's Custom Message screen.
$msg	Optional string constant, variable, or expression that contains the body of the pendant's Custom Message screen. This can accept html tags to create an html-formatted text box. If $msg is an array and msgsize > 1, it will concatenate all the elements of the array. See the example code that follows.
$f1	Optional string constant, variable, or expression that contains the label of the F1 Key of the Custom Message Screen.
$f2	Optional string constant, variable, or expression that contains the label of the F2 Key of the Custom Message Screen.
$f3	Optional string constant, variable, or expression that contains the label of the F3 Key of the Custom Message Screen.
$f4	Optional string constant, variable, or expression that contains the label of the F4 Key of the Custom Message Screen.

Details

PDNT.WRITE is used to set the screen of the T20 Pendant. This is used to create a user interface to program the pendant through eV+. The screen can be either updated with a subsequent PDNT.WRITE or cleared with a PDNT.CLEAR. While the screen is displayed, all

green keys, as well as Select Robot, are active, so that the robot can always be jogged. All other keys have no effect aside from being sent to the eV+ software.

Example: Using PDNT.WRITE with the Pendant

The following instructions:

```
$p.title = "Operator Control"
$p.msg[0] = "Select Options from buttons below"
$p.f[1] = "Apps"
$p.f[2] = "Status"
$p.f[3] = ""
$p.f[4] = ""
PDNT.WRITE $p.title, $p.msg[], $p.f[1], $p.f[2], $p.f[3], $p.f[4]
```

Create the screen:

Screen Generatate by Preceding Instructions

Tag	Example
Paragraph	This is a paragraph.
Line Break	
Horizontal Rule	<hr>
Bold Text	Bold text
Emphasized Text	Emphasized text
Italic Text	Italic text
Smaller Text	Smaller text
Important Text	Important text

Related Keywords

PDNT.NOTIFY program instruction

PDNT.CLEAR program instruction

#PHERE precision-point function

Syntax

#PHERE

Function

Return a precision-point value representing the current location of the currently selected robot.

Usage Considerations

The function #PHERE is considered to be a precision-point name. Thus, the # character must precede the function name whenever it is used.

PHERE is a reserved word in eV+ and cannot be used for a variable or program name.

Details

The PHERE real-valued function is equivalent to the program instruction HERE #pp.

Example:

The following example shows #PHERE being used to set a precision point value, in this case *#pp*.

```
SET #pp = #PHERE
```

Related Keyword

HERE program instruction

PI real-valued function

Syntax

PI

Function

Return the value of the mathematical constant pi (3.141593).

> **NOTE:** TYPE, PROMPT, and similar instructions display the result of the above example as a single-precision value. However, pi is actually stored and manipulated as a double-precision value. The LISTR monitor command displays real values to full precision.

Usage Considerations

The word "pi" cannot be used as a program name or variable name.

#PLATCH precision-point function

Syntax

#PLATCH (select**)**

Function

Return a precision-point value representing the location of the robot at the occurrence of the last external trigger or a Stop on Digital Signal.

Usage Considerations

The function name #PLATCH is considered to be a precision-point name. Thus, the # character must precede all uses of the function.

#PLATCH(0) returns information for the robot selected by the task executing the function. If the eV+ system is not configured to control a robot, use of the #PLATCH function does not generate an error because of the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select	Optional integer, expression, or real variable specifying: 0 Robot position latch of currently selected robot (default) n Robot position latch of robot n

Details

#PLATCH( ) returns a precision-point value that represents the location of the robot when the last trigger occurred. The LATCHED real-valued function should be used to determine when an external trigger has occurred and a valid location has been recorded.

Operation of the external trigger can be configured from the V+ System Configuration Editor in the Adept ACE software. For details, see the *Adept ACE User's Guide*.

Related Keywords

LATCH transformation function

LATCHED real-valued function

POS real-valued function

Syntax

POS (search_string, sub_string, start**)**

Function

Return the starting character position of a substring in a string.

Parameters

search_string	String expression to be searched for the occurrence of a substring.
sub_string	String expression containing the substring to be searched for within the search string.
start	Optional expression indicating the character position within the search string where searching is to begin.

Details

Returns the character position in **search_string** where **sub_string** begins. If the substring does not occur within the search string, a value of 0 is returned.

If **start** is provided, it indicates the character position within **search_string** where searching will begin. A value of 1 indicates the first character. If **start** is omitted or less than 1, searching begins with the first character. If start is greater than the length of **search_string**, a value of 0 is returned.

When checking for a matching substring, uppercase and lowercase letters are considered to be the same.

Examples

```
POS("file.ext", ".")     ;Returns 5
POS("file", ".")         ;Returns 0
POS("abcdefgh", "DE")    ;Returns 4
POS("1-2-3-4", "-", 5)   ;Returns 6
```

POWER system switch

Syntax

... POWER

Function

Control or monitor the status of high power.

Usage Considerations

DANGER: Do not use the POWER switch to enable power from within a program unless your system is subject to European certification. With European certification, special safety features are built-in to the system to prevent the robot from being activated without warning. See Details for additional information.

Using this switch to turn on high power is potentially dangerous when performed from a program because the robot can be activated without direct operator action. Turning on high power from the terminal can be hazardous if you do not have a clear view of the robot workspace or do not have immediate access to an Emergency Stop button.

Details

Enabling this switch is equivalent to pushing the COMP/PWR button on the pendant to turn on high power. If there is no error condition that prevents power from coming on, the enabling process proceeds to the second step, in which you must press the HIGH POWER button on the FP. (Systems not subject to European certification do not require the second step.)

Disabling this switch requests the robot to perform a controlled deceleration and power-down sequence. This sequence consists of:

1. Decelerating all robots according to the user-specified parameters. (See the following Note.)
2. Turning on the brakes.
3. Waiting for the user-specified brake-delay interval. (See the following Note.)
4. Turning off the amplifiers and power.
5. Asserting the backplane Emergency Stop signal and deasserting the High Power Enable (HPE) signal.

Note that DISABLE POWER may take an arbitrarily long time due to long deceleration times and long brake turn-on delays. (Use the ESTOP command or program instruction when you

desire an immediate shutdown.) The value of this switch can be checked at any time with the SWITCH real-valued function to determine if high power is on or off.

To disable power from a robot program without generating an error condition, the program must either be in DRY.RUN mode or DETACH the robot from program control. See the DRY.RUN switch or DETACH program instruction for details.

Example

The following program segment detaches the robot, turns high power off, and waits for you to turn high power back on.

```
DETACH                          ;Detach robot from program
DISABLE POWER                   ;Turn off power
TYPE "Press the COMP/PWR button to continue"
ATTACH                          ;Wait for power on and attach
TYPE "Robot program continuing..."
```

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

ESTOP program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

#PPOINT precision- point function

Syntax

#PPOINT (j1_value, j2_value, j3_value, j4_value, j5_value, j6_value, j7_value, j8_value, j9_value, j10_value, j11_value, j12_value**)**

Function

Return a precision-point value composed from the given components.

Usage Considerations

The #PPOINT function name is considered to be a precision-point name. Thus, the # character must precede all uses of the function.

Parameters

j1_value j2_value j3_value ... j12_value	Optional real-valued expressions for the respective robot joint positions. (If more values are specified than the number of robot joints, the extra values are ignored.)

Details

Returns a precision-point value composed from the given components, which are the positions of the first through last robot joints, respectively.

A zero value is assumed for any parameter that is omitted.

Examples

Assume that you want to perform a coordinated motion of joints 2 and 3 of a robot with 4 joints, starting from its current location. The following program segment performs such a motion:

```
   HERE #ref                    ;Define current location
   DECOMPOSE x[] = #ref         ;Fill array with components
;Move to new precision point defined with modified components
   MOVE #PPOINT(x[0], x[1]+a, x[2]-a/2, x[3])
```

The following steps lead to the same final location, but the robot joints are not moved simultaneously with this method.

```
DRIVE 2, a, 100              ;Drive joint 2
DRIVE 3, -a/2, 100           ;Drive joint 3
```

Related Keywords

DECOMPOSE program instruction

TRANS transformation function

PRIORITY real-valued function

Syntax

PRIORITY

Function

Return the current reaction lock-out priority for the program.

Usage Considerations

The name "priority" cannot be used as a program name or variable name.

This function returns the reaction lock-out priority, not the program priority of the executing program.

Details

The reaction lock-out priority for each program task is set to zero when execution of the task is initiated. The priority can be changed by the program at any time with the LOCK instruction, or the priority is set automatically when a reaction occurs as prescribed by a REACT or REACTI instruction.

The PRIORITY function can be used to determine the current setting of the reaction lock-out priority for the task executing the function.

Example

This example raises the priority, performs some operation that requires a reaction routine to be locked out, and then restores it to its previous value.

```
   save = PRIORITY         ;Save the current priority

   IF save < 10 THEN       ;Raise priority to at least 10
       LOCK 10
   END

; Access data shared by a reaction routine.
   LOCK save               ;Set priority to original value
```

Related Keywords

LOCK program instruction

REACT program instruction

REACTI program instruction

.PROGRAM program instruction

Syntax

.PROGRAM program_name(argument_list**)** ;comment

Function

Define the arguments that are passed to a program when it is invoked.

Usage Considerations

This instruction is inserted automatically by the eV+ editors when a new program is edited.

This special instruction must be the first line of every program.

The .PROGRAM statement cannot be deleted from a program.

Parameters

program_name	Name of the program in which this instruction is found.
argument_list	Optional list of variable names, separated by commas. Each variable can be any one of the data types available with eV+ (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array with all of its indexes left blank.
;comment	Optional comment that is displayed when the program is loaded from a disk file and when the DIRECTORY command is processed. (The semicolon [;] should be omitted if no comment is included.)

Details

The eV+ editors automatically enter a .PROGRAM line when you edit a new program. They also prevent you from deleting the line or changing the program name. You can, however, edit the line to add, delete, or modify the argument list. (The RENAME monitor command must be used to change the program name.)

The variables in the argument list are considered automatic variables for the named program. (See the AUTO instruction.)

When a program begins execution (for example, via an EXECUTE command or instruction or a CALL instruction), the arguments in the .PROGRAM instruction are associated with those in the EXECUTE or CALL. This association allows values to be passed between a program and its caller.

See the description of the CALL instruction for an explanation of how the program arguments receive their values from a calling program and return their values to the calling program. The following rules apply to any program argument that is omitted when the program executes:

- Real-valued scalar parameters *can* be assigned a value within a program if they are omitted.
- Location, string, and belt (scalar or array) parameters, and real-valued array parameters, *cannot* be assigned a value within a program if they are omitted. (AUTO variables can be used to work around this restriction, as shown in the example below.) However, undefined parameters can be passed as program arguments and then be assigned a value.

> **NOTE:** If a program attempts to assign a value to one of these omitted variables, the error *Undefined value* results. In that case, the error refers to the variable on the left side of the assignment instruction.

- If an undefined or omitted parameter is passed to another program through a subsequent CALL instruction, and the type of the variable is ambiguous (i.e., the type could be real-valued or location), the parameter is assumed to be real-valued.
- Elements of an omitted array parameter cannot be passed by reference in a subsequent CALL instruction.

The DEFINED real-valued function can be used within a program to check whether a program parameter is defined (meaning both: passed as a argument, and as an argument that has been assigned a value previously). The example below shows how a program can be written to accommodate undefined or omitted parameters.

A comment can be included on the .PROGRAM line, which is displayed when the program is loaded from the disk and by the DIRECTORY command.

Examples

Define a program that expects no arguments to be passed to it:

```
.PROGRAM get()
```

Define a program that expects a string-valued argument and either a location or real-valued argument (the type of the second argument is determined by its use in the program):

```
.PROGRAM test($n, dx)
```

The following program segment shows how a program can be written to deal with undefined or omitted parameters. The example shows part of the program example, which has a real-valued parameter and a string parameter.

```
.PROGRAM example(real, $string)
        AUTO $internal.var
     ; Check for undefined or omitted real-valued scalar parameter.
        IF NOT DEFINED(real) THEN           ;If parameter is undefined
```

```
            real = 1                        ;assignment of desired
         END                                ;default value is okay
      ;Check for undefined or omitted string parameter.
         IF DEFINED($string) THEN           ;If parameter is defined,
            $internal.var = $string         ;use the parameter value
         ELSE ;Otherwise,
            $internal.var = "default"       ;use default value
         END
      ;(Program continues...)
.END
```

Refer to the DEFINED function for more details and for testing nonreal arguments.

Related Keywords

CALL program instruction

CALLP program instruction

CALLS program instruction

EXECUTE monitor command

EXECUTE program instruction

PRIME monitor command

SSTEP monitor command

XSTEP monitorcommand

PROMPT program instruction

Syntax

PROMPT output_string**, variable_list**

Function

Display a string on the system terminal and wait for operator input.

Parameters

output_string	Optional string expression that is output to the system terminal. The cursor is left at the end of the string.
variable_list	A list of real-valued variables, or a single string variable, that receives the data.

Details

Displays the text of the output string on the system terminal, and waits for you to type in a line terminated by pressing the RETURN key.

The input line can be processed in either of two ways:

1. If a list of real-valued variables is specified as the variable list, the line is assumed to contain a list of numbers separated by space characters and/or commas. Each number is converted from text to its internal representation, and its value is stored in the next variable contained in the variable list. If more values are read than the number of variables specified, the extra values are ignored. If fewer values are read, the remaining variables are set to zero. If data is read that is not a number, an error occurs and program execution stops. Each PROMPT instruction should request only one value to avoid confusion and to reduce the possibility of error.
2. If a single string variable is specified as the variable list, the entire input line is stored in the string variable. The program must then process the string appropriately.

If you press the RETURN key, or press CTRL+C, an empty line is read. This results in all the real variables being set to zero, or the string variable being assigned an empty string.

If you press CTRL+Z, an end-of-file error condition results. If there is no REACTE instruction active, program execution is terminated and an error message is displayed. Thus, CTRL+Z can be a useful way to abort program execution at a PROMPT.

Examples

Consider the instruction:

```
PROMPT "Enter the number of parts: ", part.count
```

The result of executing this instruction is the display of the message

```
Enter the number of parts:
```

on the system terminal to ask you to type in the desired value. After you type a number and press the RETURN key, the variable part.count is set equal to the value typed, and program execution resumes.

Consider changing the above instruction to:

```
PROMPT "Enter the number of parts: ", $input
```

Even if you enter characters that are not valid for numeric input, eV+ does not output an error message. The application program can use the various string functions to extract numeric values from the input string.

If you want to include format specifications in the string output to the terminal (such as /Cn to skip lines), you can use either the $ENCODE function or the TYPE instruction. For example, the instruction

```
PROMPT $ENCODE(/B,/C1,/X10)+"Enter the number of parts: ", $input
```

beeps the terminal, spaces down a line, spaces over ten spaces, outputs the string, and waits for your input. (Note that a + sign has to be used between the $ENCODE function and the quoted string because the entire **output_string** parameter must be a single string expression.)

The following pairs of instructions are equivalent to the previous example:

```
TYPE /B, /C1, /X10, /S
PROMPT "Enter the number of parts: ", $input
```

or

```
TYPE /B, /C1, /X10, "Enter the number of parts: ", /S
PROMPT , $input
```

Note that /S must be included in the TYPE instructions as shown to have the prompt string output on one line, and to have the cursor remain on that line.

Related Keywords

GETC real-valued function

READ program instruction

TYPE program instruction

RANDOM real-valued function

Syntax

RANDOM

Function

Return a pseudorandom number.

Usage Considerations

The word "random" cannot be used as a program name or variable name.

Details

Returns a pseudorandom number in the range 0.0 to 1.0, inclusive. Thus, each time the RANDOM function is evaluated, it returns a different value.

The numbers generated by this function are pseudorandom because the sequence repeats after this function has been called 2^{24} (16,777,216) times.

REACT program instruction

Syntax

REACT signal_num, program, priority

Function

Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.

Usage Considerations

The REACT (and REACTI) instruction can be executed by any of the program tasks. That is, each task can have its own, independent reaction definition.

Any of the first twelve external input signals (1001 to 1012) can be simultaneously monitored.

Reactions are triggered by signal *transitions* and not levels. Thus, if a signal is going to be monitored for a transition from off to on and the signal is already on when a REACT (or REACTI) instruction is executed, then the reaction does not occur until the signal goes off and then on again.

A signal must remain stable for at least 18 milliseconds to assure detection of a transition.

> **NOTE:** If software signals are being used to trigger reactions, the WAIT instruction (with no argument) should be used as required to ensure that the signal state remains constant for the required time period.

The requested signal monitoring is enabled as soon as a REACT (or REACTI) instruction is executed. Because of the way eV+ processes program instructions, this can result in an effect on the motion initiated by the motion instruction *preceding* the REACT (or REACTI) instruction in the program. (See the section Motion Control Examples in the *eV+ Language User's Guide* for a discussion of robot motion processing.)

Parameters

signal_num	Real-valued expression representing the signal to be monitored. The signal number must be in the range 1001 to 1012 (external input signals) or 2001 to 2008 (internal software signals). (The software signals can thus be used by one program task to interrupt another task.) If the signal number is positive, eV+ looks for a transition from off to on; if **signal_num** is negative, eV+ looks for a transition from on to off.
program	Name of the subroutine that is to be called when the signal

transitions properly.

priority	Optional real-valued expression that indicates the relative importance of this reaction as explained below. The value of this expression is interpreted as an integer value and can range from 1 to 127. See the LOCK instruction for additional details on priority values. The default value is 1.

Details

When the specified signal transition is detected, eV+ reacts by checking the priority specified with the REACT instruction against the program priority setting at that time. (The program priority is always set to 0 when execution begins. It can be changed with the LOCK instruction.) If the REACT priority is greater than the program priority, the normal program execution sequence is interrupted and the equivalent of a CALL program instruction is executed. Also, the program priority is temporarily raised to the REACT priority, locking out any reactions of equal or lower importance. When a RETURN instruction is executed in a reaction subroutine, the program priority is restored to the value it had before the reaction program was invoked.

If the REACT priority is less than or equal to the program priority when the signal transition is detected, the reaction is queued and does not occur until the program priority is lowered. Therefore, depending upon the relative priorities, there can be a considerable delay between the time a signal transition is noticed by eV+ and the time the reaction program is actually invoked.

If multiple reactions are pending because of a priority lockout, the reaction with the highest priority is serviced first when the locking priority is lowered. If multiple pending reactions have the same priority, the one associated with the highest signal number is processed first.

The subroutine call to program is performed such that when a RETURN instruction is encountered, the next instruction to be executed is the one that follows the last instruction processed before the reaction program was initiated. If there is a sequence of instructions that you do not want interrupted by a reaction program, you should use the LOCK instruction to raise the program priority during that sequence.

The signal monitoring continues until one of the following occurs:

- An IGNORE instruction is executed for the signal.
- A reaction occurs (in which case IGNORE signal_num is automatically performed).
- A REACT (or REACTI) instruction is executed that refers to the same signal. That is, if the signal specified in a REACT instruction is already being monitored by a previous REACT or REACTI instruction, the old instruction is canceled when the new REACT instruction is executed.

Example

The instruction below monitors the external input signal identified by the value of the variable test. If the desired signal transition occurs (as specified by the sign of the value of test), program execution branches to program delay as soon as the program priority drops to 0 (since no priority is specified in the instruction). (The program priority is raised to 1 [the default value] when the subroutine is invoked; the program priority returns to 0 when the program returns.)

```
REACT test, delay
```

Related Keywords

IGNORE program instruction

LOCK program instruction

PRIORITY real-valued function

REACTE program instruction

REACTI program instruction

SIG.INS real-valued function

REACTE program instruction

Syntax

REACTE program_name

Function

Initiate the monitoring of errors that occur during execution of the current program task.

Usage Considerations

The main purpose for the REACTE instruction is to allow for an orderly shutdown of the system if an unexpected error occurs. If a robot hardware error occurs, for example, a REACTE program can set external output signal lines to shut down external equipment. Using the REACTE instruction for other purposes requires extreme caution.

The REACTE instruction can be executed by any of the program tasks. That is, each task can have its own, independent REACTE definition. (A task cannot directly trap errors caused by another task, but tasks can signal each other via global variables or software signals.)

The ERROR real-valued function must be called before a REACTE with no program name, since the REACTE clears the previous errors

See the list below for other considerations.

Parameter

program_name	Optional name of the program that is to be called when a program error occurs. If no program is specified, the previous REACTE is canceled, and any pending error message is discarded.

Details

If an error occurs after a REACTE instruction has been executed, the specified program is invoked, rather than stopping normal program execution. (The program is invoked as though by the CALL program instruction.) The ERROR real-valued function can be used within the error-handling program to determine what error caused the program to be invoked.

There are several special considerations that must be kept in mind when using this facility:

- The program priority is raised to 254 when the error-handling program is invoked, locking out all reaction programs.
- Execution of the program task stops if an error occurs while the system is processing a previous error.
- There must be room on the user program stack for one more subroutine. Therefore,

the error *Too many subroutine calls* cannot be processed. (See the STACK monitor command.)

- The error-handling program can contain a RETURN instruction. When it is executed, the program tries to re-execute the instruction that caused the error. Note that this may cause an endless loop if the error continues to occur.
- Before the error-handling program is entered, a DETACH instruction for the robot (logical unit number 0) is effectively executed. Thus, an ATTACH instruction must be executed for the robot before program control of the robot can resume.
- If a STOP, HALT, or PAUSE instruction is executed within the error-handling program, the original error message is output unless the error-handling program contains a REACTE instruction with no argument.
- Unlike REACT and REACTI, execution of the REACTE error-handling program is never deferred because of priority considerations.

Example

Initiate monitoring of errors so that the program error.trap is executed if any error should occur during execution of the current program task:

```
REACTE error.trap
```

Related Keywords

ERROR real-valued function

REACT program instruction

REACTI program instruction

RETURNE program instruction

REACTI program instruction

Syntax

REACTI signal_num, program**,** priority

Function

Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal transitions properly and optionally trigger a subroutine call.

Usage Considerations

For most applications, the REACTI instruction should be used only in a robot control program. (See below for more information.)

When a REACTI triggers, the robot that is stopped is the one selected by the task at the time of the trigger, regardless of which robot was selected at the time the REACTI instruction was executed.

Also see the considerations listed for the REACT program instruction.

Parameters

signal_num	Real-valued expression representing the signal to be monitored. The signal number must be in the range 1001 to 1012 (external input signals) or 2001 to 2008 (internal software signals). (The software signals can thus be used by a secondary program to interrupt the robot control program, and vice versa.) If the signal number is positive, eV+ looks for a transition from off to on; if signal is negative, eV+ looks for a transition from on to off.
program	Optional name of the subroutine that is called when the signal transitions properly.
priority	Optional real-valued expression that indicates the relative importance of this reaction as explained below. The value of this expression is interpreted as an integer value and can range from 1 to 127. If this argument is omitted, it defaults to 1. See the LOCK instruction for additional details on priority values.

Details

When the specified signal transition is detected, eV+ reacts by immediately stopping the current robot motion. If a program is specified, eV+ then continues processing the reaction just as it would for a REACT instruction. (See the description of the REACT instruction for a full explanation of this processing).

When REACTI is used by a program task that is not actually controlling the robot, care must be exercised to make sure the robot control program does not nullify the intended effect of the reaction subroutine. That is, if your application has one program task monitoring the signal and a different program task controlling the robot, you should keep the following points in mind when planning for processing of the reaction:

- The robot motion in process at the time of the reaction is stopped, as if a BRAKE instruction were executed, but execution of the robot control program is not directly affected.
- If a reaction subroutine is specified, that routine is executed by the task that is monitoring the reaction (not by the task controlling the robot).

The signal monitoring continues until one of the following occurs:

- An IGNORE instruction is executed for the signal.
- A reaction occurs (in which case IGNORE signal_num is automatically performed).
- A REACTI (or REACT) instruction is executed that refers to the same signal. That is, if the signal specified in a REACTI instruction is already being monitored by a previous REACTI or REACT instruction, the old instruction is canceled when the new REACTI instruction is executed.

If you do not want the robot motion to stop until the reaction program is actually called, you should use a REACT instruction and put a BRAKE instruction in the reaction program.

Example

The instruction below initiates monitoring of external input signal #1001. The robot motion is stopped immediately if the signal ever changes from on to off (since the signal is specified as a negative value). A branch to program alarm then occurs when the program priority falls below 10 (if it is not already at or below that level).

```
REACTI -1001, alarm, 10
```

Related Keywords

IGNORE program instruction

LOCK program instruction

PRIORITY real-valued function

REACT program instruction

REACTE program instruction

SIG.INS real-valued function

READ program instruction

Syntax

READ (lun, record_num, mode**) var_list**

Function

Read a record from an open file or from an attached device that is not file oriented. For an network device, read a string from an attached and open TCP connection.

Usage Considerations

The logical unit referenced by this instruction must have been attached previously.

For file-oriented devices, a file must already have been opened with an FOPEN_ instruction.

Parameters

lun	Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)
record_num	Optional real-valued expression that specifies the record to read for file-oriented devices opened in random-access mode (see the FOPEN_ instructions). For nonfile-oriented devices or for sequential access of a file, this parameter should be 0 or omitted. Records are numbered from one to a maximum of 16,777,216. When accessing the TCP device with a server program, this parameter is an optional real variable that *returns* the client handle number. The handle can be used to identify the client accessing a multiple-client server.
mode	Optional real-valued expression that specifies the mode of the read operation. Currently, the mode is used only for the terminal and serial I/O logical units. The value is interpreted as a sequence of bit flags as detailed below. (All bits are assumed to be clear if no mode value is specified.) Bit 1 (LSB) Wait (0) vs. No-wait (1) (mask value = 1) If this bit is clear, program execution is suspended until the read operation is completed. If the bit is set and the requested data is not available, program execution continues immediately and the IOSTAT function returns the error code for *No data received* (-526).

> **NOTE:** For a no-wait READ to access a serial line, the line must be configured to use DDCMP.

Bit 2 Echo (0) vs. No-echo (1) (mask value = 2)

If this bit is clear, input from the terminal is echoed back to the source. If the bit is set, characters are not echoed back to the source. (This mode bit is ignored for the serial lines.)

var_list Either a list of real-valued input variables or a list of string variables, which receives the data (see following details).

Details

This is a general-purpose data input instruction that reads a record from a specified logical unit. A record can contain an arbitrary list of characters but must not exceed 512 characters in length. For files that are opened in fixed-length record mode, this instruction continues to read characters until it has read exactly the number of characters specified during the corresponding FOPEN_ instruction.

For variable-length record mode (with most devices), this instruction reads characters until the first carriage-return (CR) and line-feed (LF) character sequence (or Ctrl+Z) is encountered. Thus, for example, if you perform a variable-length record mode read from the disk, you receive all the characters until a CR and LF are encountered.

The special character Ctrl+Z (26 decimal) indicates the logical end of the file, which is reported as an error by the IOSTAT function. No input characters can be read beyond that point.

READ operations from the terminal, the pendant, and the serial lines are always assumed to be in variable-length record mode. Except as noted below, the records are terminated by CR and LF (which are not returned as part of the record). Thus, a READ from these devices is not complete until a CR and LF are received as input. For example, if you perform a READ from the terminal, you receive all the characters until the RETURN key is pressed.

> **NOTE:**When a CR is received from the system terminal, eV+ automatically adds a LF. Similarly, the pendant's DONE key is interpreted as CR and LF.
>
> The GETC real-valued function can be used instead of the READ instruction if you want to receive the CR and LF characters at the end of a record.
>
> When a READ instruction accesses a serial line configured to use DDCMP, the record may contain arbitrary data, including CR and LF characters.

If bit 1 is set in the mode value, a read operation that is not complete does not cause the program to wait, but returns immediately with the error *No data received* (error code -

526). Then, additional READ instructions must be executed, until one is complete, in order to obtain the data in the variable list. The IOSTAT function can be used to determine when such a READ is complete.

Once a record has been read, it is processed in one of the following two ways:

1. If the **var_list** parameter is a list of real-valued variables, the record is assumed to contain a list of numbers separated by space characters and/or commas. Each number is converted from text to its internal representation, and its value is stored in the next variable contained in the variable list. If more values are read than the number of variables specified, the extra values are ignored. If fewer values are read, the remaining variables are set to zero. If data is read that is not a number, an error occurs and program execution stops (or an error reaction occurs).
2. If the **var_list** parameter is a list of string variables, the entire record is stored in the string variables as follows. The first 128 bytes in the record are copied to the first string variable. If there are more than 128 bytes in the record, the second string variable is filled with the next 128 bytes. This continues until the entire record has been processed or all the string variables have been filled.

   If there is not enough data to fill all the string variables, the unused string variables are set to the empty string (""). If there is too much data for the number of string variables specified, an error is reported by the IOSTAT real-valued function.

   When a READ is performed in variable-length record mode, the strings contain all the characters up to, but not including, the terminating CR and LF, which are discarded.

Any error in the specification of this instruction (such as attempting to read from an invalid unit) causes a program error and halts program execution. However, errors associated with performing the actual read operation (such as end of file or device not ready) do not halt program execution since these errors may occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function after performing the read. In general, it is good practice always to test whether each read operation completed successfully by testing the value from IOSTAT.

When accessing a network device, the record_num parameter allows a server to communicate with multiple clients on a single logical unit. In this context, the parameter provides a*handle* number that you can use to identify the client from which the READ data was received. Handles are allocated when a client connects to the server and are deallocated when the client disconnects. In order to determine when the client connection or disconnection is done, you must use the IOSTAT real-valued function after the READ. Refer to the documentation for IOSTAT.

The READ instruction with TCP/IP reads data until either the input string is full or the buffer is empty, at which point the instruction returns. READ with TCP/IP does not allow fixed-length records and does not terminate when encountering a delimiter.

Example

Read a line of text from the disk and store the record in the string variable $disk.input:

```
READ (5) $disk.input
```

For an example of using the READ instruction with the TCP device, refer to the Example section for the IOSTAT real-valued function.

Related Keywords

ATTACH program instruction

FOPEN_ program instruction

FSEEK program instruction

GETC real-valued function

IOSTAT real-valued function

PROMPT program instruction

READY program instruction

Syntax

READY

Function

Move the robot to the READY location above the workspace, which forces the robot into a standard configuration.

Usage Considerations

Before executing this instruction with the DO monitor command (DO READY), make sure that the robot will not strike anything while moving to the READY location.

The READY instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the READY instruction causes an error.

Details

This instruction always succeeds, regardless of where the robot is located at the time.

An Adept Cobra robot has the following configuration when it is at the READY location:

- Joint 2 is close to 90 degrees
- The axis of joint 3 is in the World X-Z plane (that is, Y = 0)
- The alignment keyway in the end-effector flange is directed along the positive X axis (that is, the tool X axis is parallel to the world X axis)

The following table lists the joint positions for the READY locations for various Adept robots.

Joint	Adept Cobra s350	Adept Cobra s600	Adept Cobra s800	Adept Quattro s650/s80-0	Adept Viper s650/s85-0 /s1300	Adept Viper s1700/ s1700-D
1	-61.2°	-43.5°	-42.9°	0°	0°	0°
2	90.3°	96.8°	93.4°	0°	-90°	-90°
3	10.0 mm	10.0 mm	10.0 mm	0°	180°	180°

Joint	Adept Cobra s350	Adept Cobra s600	Adept Cobra s800	Adept Quattro s650/s80-0	Adept Viper s650/s85-0 /s1300	Adept Viper s1700/ s1700-D
4	29.1°	53.8°	50.5°	0°	0°	0°
5	N/A	N/A	N/A	N/A	0°	90°
6	N/A	N/A	N/A	N/A	0°	0°

Related Keyword

SELECT program instruction

SELECT real-valued function

RELAX and RELAXI program instruction

Syntax

RELAX

RELAXI

Function

Limp the pneumatic hand.

Usage Considerations

RELAX causes the hand to limp during the next robot motion.

RELAXI causes a BREAK in the current continuous-path motion and causes the hand to limp immediately after the current motion completes.

The RELAX instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The RELAXI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions turn off both the open and close pneumatic control solenoid valves, causing the pneumatic hand to become limp. If the RELAX instruction is used, the signal is sent when the next robot motion begins.

The RELAXI instruction differs from RELAX in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signals are sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Related Keywords

CLOSE program instruction

CLOSEI program instruction

HAND.TIME system parameter

OPEN program instruction

OPENI program instruction

SELECT program instruction

SELECT real-valued function

RELAX and RELAXI program instruction

Syntax

RELAX

RELAXI

Function

Limp the pneumatic hand.

Usage Considerations

RELAX causes the hand to limp during the next robot motion.

RELAXI causes a BREAK in the current continuous-path motion and causes the hand to limp immediately after the current motion completes.

The RELAX instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The RELAXI instruction can be executed by any program task as long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing these instructions causes an error.

Details

These instructions turn off both the open and close pneumatic control solenoid valves, causing the pneumatic hand to become limp. If the RELAX instruction is used, the signal is sent when the next robot motion begins.

The RELAXI instruction differs from RELAX in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signals are sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

Related Keywords

CLOSE program instruction

CLOSEI program instruction

HAND.TIME system parameter

OPEN program instruction

OPENI program instruction

SELECT program instruction

SELECT real-valued function

RELEASE program instruction

Syntax

RELEASE

Function

Allow the next available program task to run.

Details

This instruction releases control to another task that is ready to run. For more information on task scheduling, see the section Scheduling of Program Execution Tasks in the *eV+ Language User's Guide*.

This instruction can be used in place of the WAIT instruction (with no arguments) in cases where other tasks must be given an opportunity to run, but a delay until the next trajectory cycle is not desired.

Related Keywords

WAIT program instruction

WAIT.EVENT program instruction

RESET program instruction

Syntax

RESET

Function

Turn off all the external output signals.

Details

The RESET program instruction is useful in the initialization portion of a program to ensure that all the external output signals are in a known state.

DANGER: Before issuing this instruction, make sure all devices connected to the output signals can safely be turned off. Be especially careful of signals that start an action when they are turned off.

Related Keywords

BITS monitor command

BITS program instruction

BITS real-valued function

IO monitor command

RESET monitor command

SIG real-valued function

SIG.INS real-valued function

SIGNAL monitor command

SIGNAL program instruction

RETURN program instruction

Syntax

RETURN

Function

Terminate execution of the current subroutine, and resume execution of the suspended program at its next step. A program may have been suspended by issuing a CALL, CALLP, or CALLS instruction, or by the triggering of a REACT, REACTE, or REACTI condition.

Details

A RETURN instruction in a main program has the same effect as a STOP instruction.

A RETURN instruction is assumed if program execution reaches the last step of a subroutine. However, it is not good programming style to use this feature-an explicit RETURN instruction should be included as the last line of each subroutine.

The effect of a RETURN instruction in an error reaction subroutine differs slightly. In that case, if the reaction subroutine was invoked because of a program error (as opposed to an asynchronous servo error or PANIC button press), the statement that caused the error is executed again. That is, the error may occur again immediately. The RETURNE instruction should be used in error reaction subroutines to avoid that situation.

If a RETURN instruction is used to exit from a reaction routine, the program reaction priority is restored to whatever it was before the reaction routine started execution.

Related Keywords

CALL program instruction

CALLP program instruction

CALLS program instruction

LOCK program instruction

REACT program instruction

REACTE program instruction

REACTI program instruction

RETURNE program instruction

RETURNE program instruction

Syntax

RETURNE

Function

Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the instruction that caused the subroutine to be invoked.

Details

The RETURNE instruction is intended for use in error reaction subroutines. That is, subroutines that are invoked, through the REACTE mechanism, as a result of an error during program execution.

If a RETURNE instruction is used to exit from an error reaction routine, the program reaction priority is restored to whatever it was before the error reaction routine started execution.

When a RETURNE instruction is executed in an error reaction subroutine, then execution continues with the statement following the one executing when the error occurred. (Note that in this situation, a RETURN instruction results in the statement that generated the error being executed again, possibly causing an immediate repeat of the error.)

> **NOTE:** Because of the forward processing ability of eV+, the instruction that is the source of an error may not be the one executing when the error is actually registered. For example, when a MOVE instruction is processed, the robot begins moving, but during the motion several additional instructions may be processed. If an envelope or similar error occurs after this forward processing, the RETURNE is based on the instruction processing when the error occurs, not the MOVE instruction.

It may be helpful to note that the RETURNE instruction behaves similarly to the PROCEED command. The RETURN instruction behaves similarly to the RETRY command (except that with RETURN an interrupted robot motion is not restarted).

A RETURNE instruction in a program that is *not* executed in response to an error has the same effect as a RETURN instruction. RETURNE, however, takes slightly longer to execute than does RETURN.

Related Keywords

REACTE program instruction

RETURN program instruction

RIGHTY program instruction

Syntax

RIGHTY

Function

Request a change in the robot configuration during the next motion so that the first two links of the robot resemble a human's right arm.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a right-handed configuration, this instruction is ignored by the robot.

The RIGHTY instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the RIGHTY instruction causes an error.

See LEFTY/RIGHTY.

Related Keywords

CONFIG real-valued function

LEFTY program instruction

SELECT program instruction

SELECT real-valued function

ROBOT system switch

Syntax

... ROBOT [index]

Function

Enable or disable one robot or all robots.

Usage Considerations

The ROBOT system switches may be modified only when both of the following conditions are satisfied:

1. The POWER system switch is OFF.
2. When the eV+ system was booted from disk, at least one robot started up without reporting a fatal error.

The maximum number of robots supported depends on your controller configuration.

Some controllers do not allow a robot to be calibrated unless all robots with lower index numbers are enabled.

Parameter

index	Optional real value, variable, or expression (interpreted as an integer) that specifies the robot to be enabled or disabled. The value should be 1 through 15 (corresponding to robots 1 through 15, respectively). If the index is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

When the eV+ system starts up (after booting from disk), all the robots that started up without reporting a fatal error are enabled by default, and all the corresponding ROBOT switches are enabled. After start up, the ROBOT switches can be used to selectively disable robots. For example, this can aid in the debugging of individual robots.

The ROBOT switches may be modified only for robots that are present and that started up without a fatal error.

When a robot is disabled by use of the ROBOT switch, that robot is bypassed when:

- Power is enabled for all robots with the COMP/PWR button on the pendant or with the POWER system switch.

- All the robots are calibrated via the CALIBRATE monitor command or program instruction.

Motion instructions should not be executed for a robot that has been disabled.

The settings of these switches can be checked at any time with the SWITCH monitor command or real-valued function to determine which robots are enabled.

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

ROBOT.OPR program instruction

Syntax

ROBOT.OPR (function_code) exp1, exp2, ..., expn

Function

Execute operations that are specific to the currently selected robot or robot module.

Usage Considerations

ROBOT.OPR is a general-purpose instruction whose interpretation varies from one robot type to another.

The instruction is ignored if there is no robot attached.

An error is reported if the robot is tracking a belt or if ALTER is active.

Parameters

function_code	Optional real value that specifies a function for the selected robot module.
exp1, exp2, ..., expn	Optional expressions whose interpretation is determined by the selected robot module.

NOTE: An *Invalid argument* error is returned if the "function_code" value used is not valid. The same error is returned if an "exp" value exceeds its allowable range. The valid "function_code" and "exp" values are described in the device module documentation for your robot. See the Adept Robot Device Modules menu in the Adept Document Library to access the device module for your robot.

Details

This instruction executes operations that are specific to the currently selected robot or robot module. If the selected robot does not support any special operations, this instruction has no effect.

The following table shows the "function" and "exp" values for the supported robot modules. For additional details about the applicability and use of this instruction, refer to the documentation for your specific robot module. See the Adept Robot Device Modules menu to access the documentation for the device module for your robot.

NOTE: Only device modules for non-Adept robots are documented.

Supported Robot Modules

Robot Module	function_ code	exp1, exp2,...ex-pn	Description
XY3 (module ID 21)	When the system is first booted, the first Z/Theta pair is selected as the primary head, and the second and third heads are not slaved. The ROBOT.OPR program instruction must be executed to specify a different primary head, and to slave heads. The following points should be kept in mind when utilizing ROBOT.OPR to change the head selection: 1. This instruction breaks any executing continuous-path motion and modifies the head selection after the robot has come to a stop. 2. Before you slave one head to another head, you should ensure that the Z and Theta values for the slave head are identical to those of the primary head. If any differences exist, at the start of the next motion, the axes of the slave head will quickly jump to the same joint positions as the primary head.		
	0 enables selection of the primary and slaved axes	exp1	Number of the primary Z/Theta axes (1-N).
		exp2	(Optional) Number of the secondary Z/Theta axes (1-N).
		exp3	(Optional) Number of the third Z/Theta axes (1-N).
		exp4	(Optional) Number of the fourth Z/Theta axes (1-N).
		NOTE: "N" is the number of configured Z/Theta pairs, which can be 1 to 4. There is no check for an axis pair being specified more than once.	
	1 enables definition of the position-offset values	exp1	Z offset for first slaved axes.

Robot Module	function_ code	exp1, exp2,...exp-pn	Description
	that are added to the commanded Z and RZ positions for each slave axis		
		exp2	(Optional) RZ offset for first slaved axes.
		exp3	(Optional) Z offset for second slaved axes.
		exp4	(Optional) RZ offset for second slaved axes.
		exp5	(Optional) Z offset for third slaved axes.
		exp6	(Optional) RZ offset for third slaved axes.
		NOTE: The offsets for the fourth set of slaved axes cannot be set.	
DLT (module ID 27)	For this robot, this instruction sets the Cartesian acceleration parameters to one of three sets of values. The purpose of this operation is to adjust the dynamic performance of the robot depending upon the payload being carried. **NOTE:** When utilizing ROBOT.OPR to change the acceleration values, the robot must be attached and stopped when this instruction is executed, to ensure no adverse interactions with any current motion execution.		

<table>
<tr><th>Robot Module</th><th>function_ code</th><th>exp1, exp2,...ex-pn</th><th>Description</th></tr>
<tr><td></td><td>0 sets the Cartesian acceleration parameters to one of three sets of values.</td><td>exp1</td><td>Number of the set of Cartesian acceleration parameters to utilize (1-3). The parameters are intended to be used as follows:<table><tr><th>Parameter Set Number</th><th>Intended Payload</th></tr><tr><td>1</td><td>0-1 kg</td></tr><tr><td>2</td><td>1-3 kg</td></tr><tr><td>3</td><td>3-5 kg</td></tr></table></td></tr>
<tr><td rowspan="2">Adept Quattro s650 Robot</td><td colspan="3">For this robot, this instruction sets the Cartesian acceleration parameters to one of two sets of values. The purpose of this operation is to adjust the dynamic performance of the robot depending upon the payload being carried.
NOTE: When utilizing ROBOT.OPR to change the acceleration values, the robot must be attached and stopped when this instruction is executed, to ensure no adverse interactions with any current motion execution.</td></tr>
<tr><td>0 (must be set to 0)</td><td>exp1</td><td>Number of the set of Cartesian acceleration parameters to utilize (1 or 2). (Parameter set #1 is applied when the eV+ system is booted from disk.)</td></tr>
</table>

<table>
<tr><th>Robot Module</th><th>function_ code</th><th>exp1, exp2,...exp-n</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Parameter Set Number</th><th>Intended Payload</th><th>Notes</th></tr>
<tr><td>1</td><td>Default</td><td rowspan="2">Parameter sets 1-7 are available with all Adept Quattro robots.</td></tr>
<tr><td>2</td><td>0-1 kg</td></tr>
<tr><td>3</td><td>1-2 kg</td><td rowspan="5">NOTE: Payloads greater than 6 kg with an Adept Quattro s650 robot, or 4 kg with an Adept Quattro s800 robot, should be used only with the P30 (fixed) platform.</td></tr>
<tr><td>4</td><td>2-4 kg</td></tr>
<tr><td>5</td><td>4-6 kg</td></tr>
<tr><td>6</td><td>6-8 kg</td></tr>
<tr><td>7</td><td>8-10 kg</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Robot Module</th><th>function_ code</th><th>exp1, exp2,...ex-pn</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Parame-ter Set Number</th><th>Intende-d Payload</th><th>Notes</th></tr>
<tr><td>8</td><td>10-12 kg</td><td rowspan="2">These selections are available only with the Adept Quattro s650H/s65-0HS robots.</td></tr>
<tr><td>9</td><td>12-15 kg</td></tr>
</table>
</td></tr>
</table>

Related Keyword

ROBOT.OPR real-valued function

ROBOT.OPR real-valued function

Syntax

ROBOT.OPR (mode, index)

Function

Returns robot-specific data for the currently selected robot.

Parameters

mode, index	The values of these two parameters select what robot-specific data is returned.

Details

The following table shows the mode and index information for the supported robot modules. For additional details about the applicability and use of this instruction, refer to the documentation for your specific robot module. See the Adept Robot Device Modules menu to access the documentation for the device module for your robot.

> **NOTE:** Only device modules for non-Adept robots are documented.

Supported Robot Modules

Robot Module	Mode	Indices	Description
XY3 (module ID 21)	1	1 - 5	The number of the primary ZTheta pair, followed by the number of the first slave pair, the second slave pair, and the third slave pair. The first zero value indicates the end of the list of active slave pairs.
	2	1, 2	Always 0
		3	Z offset for the first slaved ZTheta
		4	RZ offset for the first slaved ZTheta
		5	Z offset for the second slaved ZTheta

Robot Module	Mode	Indices	Description
		6	RZ offset for the second slaved ZTheta
		7	Z offset for the third slaved ZTheta
		8	RZ offset for the third slaved ZTheta
Adept Quattro s650 Robot	1	1	Number of the set of acceleration parameters that is in effect. The value 0 indicates no set has been explicitly selected, and the default parameters (set #1) are in effect.
		2	Number of sets of acceleration parameters that are available for the current robot.
	2	1	Maximum tool-flange rotation angle (i.e., maximum deflection from 0 degrees).
		2	Size of the tool-flange rotation "ambiguity" zone at each end of the range of tool-flange rotation.

Related Keywords

ROBOT.OPR program instruction

SELECT program instruction

SELECT real-valued function

RUNSIG program instruction

Syntax

RUNSIG signal_num

Function

Turn on (or off) the specified digital signal as long as execution of the invoking program task continues.

Usage Considerations

Only one RUNSIG signal can be in effect for each program task.

Parameter

signal_num	Optional real-valued expression that specifies one of the digital output signals (or an internal software signal) that is to be controlled. The signal is set to on during program execution if the value is positive. A negative value results in the signal being set to off during program execution, and turned on when execution stops. If no signal is specified, any RUNSIG in effect for the task is canceled.

Details

This instruction causes the specified digital signal to be turned on (or off) as soon as the instruction is executed. The signal is turned off (or on) as soon as execution of the invoking program task stops (or the STOP instruction is executed).

This instruction is useful in an application where auxiliary equipment must be stopped if an error occurs during program execution.

Only one signal can be activated by a RUNSIG instruction at any one time (for each program task). An error condition results unless a program cancels the first RUNSIG before attempting to initiate a second.

If program execution is interrupted after a RUNSIG instruction has been executed, the specified signal returns to the selected state again if a PROCEED or RETRY command is issued. If an SSTEP or XSTEP command is issued, the signal returns to the specified state during execution of the instruction that is invoked. Similarly, processing of a DO command temporarily activates the RUNSIG signal for the corresponding program task. (The EXECUTE command and instruction cancel any previous RUNSIG for the specified program task.)

Example

Turn on the digital signal identified by the value of the variable run.signal (assuming the value is positive):

```
RUNSIG run.signal
```

The signal remains on throughout execution of the current program. The signal goes off when execution ends.

Related Keywords

IO monitor command

RESET monitor command

SIG real-valued function

SIG.INS real-valued function

SIGNAL monitor command

SIGNAL program instruction

RX, RY, RZ transformation functions

Syntax

RX (angle)

RY (angle)

RZ (angle)

Function

Return a transformation describing a rotation.

Parameter

angle	Real-valued expression that represents the rotation angle in degrees.

Details

These functions generate a transformation whose value consists of a rotation about the axis associated with the function name and a zero displacement
(X, Y, Z = 0).

Example

Produce a transformation that describes a pure 30-degree rotation about the World X axis:

```
RX(30)
```

Related Keyword

DX real-valued function

DY real-valued function

DZ real-valued function

RX, RY, RZ transformation functions

Syntax

RX (angle)

RY (angle)

RZ (angle)

Function

Return a transformation describing a rotation.

Parameter

angle	Real-valued expression that represents the rotation angle in degrees.

Details

These functions generate a transformation whose value consists of a rotation about the axis associated with the function name and a zero displacement
(X, Y, Z = 0).

Example

Produce a transformation that describes a pure 30-degree rotation about the World X axis:

```
RX(30)
```

Related Keyword

DX real-valued function

DY real-valued function

DZ real-valued function

RX, RY, RZ transformation functions

Syntax

RX (angle)

RY (angle)

RZ (angle)

Function

Return a transformation describing a rotation.

Parameter

angle	Real-valued expression that represents the rotation angle in degrees.

Details

These functions generate a transformation whose value consists of a rotation about the axis associated with the function name and a zero displacement
(X, Y, Z = 0).

Example

Produce a transformation that describes a pure 30-degree rotation about the World X axis:

```
RX(30)
```

Related Keyword

DX real-valued function

DY real-valued function

DZ real-valued function

SCALE transformation function

Syntax

SCALE (transformation BY scale_factor)

Function

Return a transformation value equal to the transformation parameter with the position scaled by the scale factor.

Parameters

transformation	Transformation expression that is to be scaled.
scale_factor	Real-valued expression that is used to scale the transformation parameter value.

Details

The value returned is equal to the value of the input transformation parameter value except that the X, Y, and Z position components are multiplied by the scale factor parameter. The rotation components have their values unchanged.

Example

If the transformation x has the value:

```
(200, 150, 100, 10, 20, 30)
```

then executing the instruction:

```
SET y = SCALE(x BY 1.25)
```

results in the transformation y receiving the value:

```
(250, 187.5, 125, 10, 20, 30)
```

Related Keyword

SHIFT transformation function

SCALE.ACCEL system switch

Syntax

... SCALE.ACCEL [robot_num]

Function

Enable or disable the scaling of acceleration and deceleration as a function of program speed, as long as the program speed is below a preset threshold.

Parameter

robot_num	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

This switch is enabled when the eV+ system is initialized.

If robot_num is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected. If robot_num is omitted or zero when the switch is accessed with the SWITCH real-valued function, the setting of the switch for robot #1 is returned.

When this switch is enabled and the program speed is below the preset threshold value, the effective acceleration and deceleration for that robot are calculated as follows:

```
effective acceleration = program_speed * acceleration_setting
effective deceleration = program_speed * deceleration_setting
```

where acceleration_setting and deceleration_setting are values set by the ACCEL instruction.

For example, if program speed 50% is specified and the threshold value is 150, the effective acceleration and deceleration are 50% of the current settings. If the program speed is higher than 150% with the threshold set to 150, the current acceleration and deceleration are used without modification.

All robot modules have the SCALE.ACCEL speed threshold set by default to a very large value, effectively forcing the scaling of accelerations and deceleration for all speeds when this switch is enabled.

CAUTION: For program speeds over 100%, if the default setting for the SCALE.ACCEL limit is used and SCALE.ACCEL is enabled, the robot is driven at much higher rates of acceleration and deceleration, as compared to V+ 11.0.

If the SCALE.ACCEL switch is disabled for a robot, accelerations and decelerations are not scaled based on the program speed. In this case, accelerations and decelerations are higher than normal at reduced speeds. This is particularly noticeable at very slow speeds. As a result, robot motions may appear to be more rough or jerky.

Example

Turn off acceleration scaling for robot #2:

```
DISABLE SCALE.ACCEL[2]
```

Related Keywords

ACCEL program instruction

ACCEL real-valued function

SPEED monitor command

SPEED program instruction

SCALE.ACCEL.ROT system switch

SCALE.ACCEL.ROT system switch

Syntax

... SCALE.ACCEL.ROT [robot_num]

Function

Specify whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.

Parameter

robot_num	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

If SCALE.ACCEL.ROT is enabled for a selected robot, the lesser of the Cartesian linear and rotational speeds is used to scale acceleration and deceleration during straight-line motions. If SCALE.ACCEL.ROT is disabled for a selected robot, only the Cartesian linear speed is considered when SCALE.ACCEL is in effect. The SCALE.ACCEL.ROT switch is enabled for all robots by default when the eV+ system is initialized.

Example

Cause SCALE.ACCEL not to use Cartesian rotational speed for robot #2:

```
DISABLE SCALE.ACCEL.ROT[2]
```

Related Keywords

ACCEL program instruction

ACCEL real-valued function

SCALE.ACCEL system switch

SPEED monitor command

SPEED program instruction

SELECT program instruction

Syntax

SELECT device_type = unit

Function

Select a unit of the named device for access by the current task.

Usage Considerations

The SELECT instruction needs to be used only if there are multiple devices of the same type connected to your system controller. This option is available only if your Adept system is equipped with the eV+ Extensions option.

The SELECT instruction affects only the task in which the instruction is executed.

The instruction SELECT ROBOT can be executed only if there is no robot attached to the current task. (If there is any doubt about whether or not a robot is attached, a program should execute a DETACH instruction before executing the SELECT instruction.)

Parameters

device_type	Keyword that identifies the type of device that is to be selected. Valid device types are ROBOT, VISION, and FORCE (which must be specified without quotation marks). The device-type keyword can be abbreviated.
unit	Real value, variable, or expression (interpreted as an integer) that specifies the particular unit to be selected. The values that are accepted depend on the configuration of the system.

Details

SELECT ROBOT

In a multiple-robot system, this program instruction selects the robot with which the current task is to communicate. (The SELECT monitor command specifies which robot the eV+ monitor is to access.) The program instruction specifies which robot receives motion instructions (for example, APPROACH and MOVE) and returns robot-related information (for example, for the HERE function).

Each time a program task begins execution, robot #1 is automatically selected. If a robot is selected, information about the robot (for example, its current position) can be accessed. In order for a program to move a robot, however, the robot must be selected and attached (with the ATTACH instruction).

As an example, if robot #2 is selected by a SELECT instruction, all motion instructions executed by the current task are directed to that robot (until another SELECT instruction is issued). Also, all robot-related functions (such as HERE) return information about robot #2.

> **NOTE:** As a convenience, when task #0 is executed, robot #1 is automatically selected and attached when program execution begins.

In order for any task to change its selected robot, no robot can be attached by the task. More than one task can have a particular robot selected, but only one task can have a robot attached. If a robot is already attached to a different task, an ATTACH waits or generates an error (depending on the mode parameter for the ATTACH instruction).

SELECT VISION

In a system with multiple vision systems, this instruction selects the vision system with which the current task is to communicate. (The SELECT monitor command specifies which vision system the eV+ monitor is to access.) This program instruction specifies which vision system receives vision instructions (for example, ENABLE VISION) and also which system returns vision-related information (for example, from the VSTATUS function).

The vision system currently selected by the monitor is automatically selected when a program begins execution.

SELECT FORCE

In a system with multiple force sensors, this monitor command or program instruction selects the force sensor with which the current task is to communicate. The SELECT monitor command specifies which force sensor the eV+ monitor is to access. The program instruction specifies which force sensor receives force instructions (for example, FORCE.READ) and returns force sensor-related information (for example, for the LATCH function).

Each time a program task begins execution, force sensor #1 is automatically selected.

Example

SELECT ROBOT Example Program

The following program selects robot #3 and moves it. This program is normally not executed by task #0, since that task is attached to robot #1 by default.

```
.PROGRAM test()
        SELECT ROBOT = 3           ;Select robot 3
        ATTACH (0,1)               ;Get control of robot 3 without waiting
        IF IOSTAT(0) < 0 THEN
            TYPE /B, "Error attaching robot: ", $ERROR(IOSTAT(0))
            PAUSE
         END
         MOVE x                    ;Move robot 3 to location "x"
         MOVE y                    ;Move robot 3 to location "y"
```

```
         DETACH                    ;Detach robot 3
.END
```

SELECT VISION Example Program

The following program segment selects vision system #2 and accesses that system.

```
.PROGRAM vision.2()
        SELECT VISION = 2          ;Select vision system #2
        ENABLE VISION              ;Enable that vision system
        VSTATUS(1,0) status[]      ;Get status information
        IF status[0] == 0 THEN     ;If vision system is idle,
            VPICTURE(1)            ;take a picture
        END
.END
```

SELECT FORCE Example Program

The following program selects force sensor #2 and reads the current forces from it.

```
.PROGRAM test()
        SELECT FORCE = 2           ;Select force sensor 2
        FORCE.READ f[] ;Read sensor 2 forces
        TYPE "Current forces on sensor", SELECT(FORCE), /S
        TYPE  "are ", /F0.1, f[0], /X1, f[1], /X1, f[2]
.END
```

Related Keywords

ATTACH program instruction

SELECT real-valued function

SELECT real-valued function

Syntax

SELECT (device_type, mode)

Function

Return the unit number that is currently selected by the current task for the device named.

Parameters

device_type	Keyword that identifies the type of device that is to be selected. The only valid type is ROBOT. The device-type keyword can be abbreviated.
mode	Optional real value, variable, or expression (interpreted as an integer) that specifies the mode for the function. If this parameter is omitted or has the value 0, the function returns the number of the unit currently selected, or 0 if no unit is selected. If mode has the value -1, the function returns the total number of units available for the specified device.

Details

This function returns either the number of the specified device that is currently selected, or the total number of devices connected to the system controller. Multiple devices of the same type are supported only if your system includes the optional eV+ Extensions software.

If the eV+ system is not configured to control a robot, the selected robot is always #1, and the total number of robots is zero.

SELECT(ROBOT) returns the number of the currently selected robot. SELECT(ROBOT,-1) returns the maximum robot number in a eV+ system.

Examples

Return the unit number of the robot selected for the current task:

```
our.robot = SELECT(ROBOT)
```

Return the total number of robots connected to the controller:

```
num.robots = SELECT(ROBOT,-1)
```

Related Keywords

SELECT monitor command

SELECT program instruction

SET program instruction

Syntax

SET location_var = location_value

Function

Set the value of the location variable on the left equal to the location value on the right of the equal sign.

Parameters

location_var	Single location variable or compound transformation that ends with a transformation variable.
location_value	Location value of the same type as the location variable on the left of the equal sign, defined by a variable or function (or compound transformation).

Details

An error message is generated if the right-hand side is not defined or is not the same type of location representation (that is, transformation or precision point).

If a compound transformation is specified to the left of the equal sign, only its right-most relative transformation is defined. An error condition results if any other transformation in the compound transformation is not already defined.

If a transformation variable is specified on the left-hand side, the right-hand side can contain a transformation, a compound transformation, or a transformation function.

Examples

Set the value of the transformation pick equal to the location of corner plus the location of shift relative to corner:

```
SET pick = corner:shift
```

Set the value of the precision point #place equal to that of the precision point #post:

```
SET #place = #post
```

Set the value of the transformation part to the current robot location, relative to the transformation pallet.

```
SET pallet:part = HERE
```

Set the value of loc1 to X = 550,Y = 450, Z = 750, y = 0, p = 180, r = 45:

```
SET loc1 = TRANS(550, 450, 750, 0, 180, 45)
```

Related Keywords

HERE monitor command

HERE program instruction

SET.EVENT program instruction

Syntax

SET.EVENT task, flag

Function

Set an event associated with the specified task.

Parameters

task	Optional real value, variable, or expression (interpreted as an integer) that specifies the task for which the event is to be set. The valid range is 0 to 27, inclusive. If this parameter is omitted, the number of the current task is used.
flag	Not used, defaults to 1.

> **NOTE:** All 28 tasks are available only in systems equipped with the optional V+ Extension.

Details

This instruction sets the event associated with the specified task. For example, if a task had been suspended by a WAIT.EVENT 1 instruction, executing the SET.EVENT instruction for that task causes it to resume execution (during the next available time slice for which it is eligible).

Related Keywords

CLEAR.EVENT program instruction

GET.EVENT real-valued function

WAIT.EVENT program instruction

#SET.POINT precision point function

Syntax

#SET.POINT

Function

Return the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.

Usage Considerations

The name "set.point" cannot be used as a program or variable name.

Details

For each trajectory-evaluation cycle, joint-angle positions are computed, converted to encoder counts, and sent to the servos as the commanded motor positions. You can use this function to capture these positions.

SETBELT program instruction

Syntax

SETBELT %belt_var = expression

Function

Set the encoder offset of the specified belt variable equal to the value of the expression.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The BELT switch must be enabled for this instruction to be executed.

The SETBELT instruction is generally used in conjunction with the BELT real-valued function to set the effective belt position to zero. (See Details section and WARNING below.)

SETBELT cannot be executed while the robot is moving relative to the specified belt variable.

The belt variable referenced must have been defined already using a DEFBELT instruction.

Parameters

%belt_var	Name of belt variable associated with the encoder offset to be set.
expression	Real-valued expression that specifies a signed 24-bit encoder offset value.

Details

When computing the position of a belt associated with a belt variable, eV+ subtracts the offset value from the current belt position value and uses the difference, modulo 16,777,216.

The expression value is normally a signed number in the range -8,388,608 to 8,388,607. If the number is outside this range, its value modulo 16,777,216 is used.

The SETBELT instruction is generally used in conjunction with the BELT real-valued function to set the effective belt position to zero. This must be done each time the robot will perform a sequence of motions relative to the belt, and must be done shortly before the first motion of such a sequence.

WARNING: It is important to execute SETBELT each time the robot is going to track the belt, to make sure the difference between the current belt position (as returned by the BELT function) and the belt position of the specified belt variable does not exceed 8,388,607 (^H7FFFFF)

during active belt tracking. Unpredictable robot motion may result if the difference does exceed this value while tracking the belt.

The SETBELT instruction can be used to synchronize robot motion with the encoder value latched by an external signal or by the AdeptVision system. See the LATCHED real-valued function and the DEVICE real-valued function for more information.

Example

The following example waits for a digital signal and then sets the belt position to zero. That is done by setting the belt offset equal to the current belt position. Finally, the robot is moved onto the belt.

```
WAIT sig(1001)
SETBELT %belt1 = BELT(%belt1)
MOVES %belt1:pickup
```

Related Keywords

BELT real-valued function

BELT system switch

DEFBELT program instruction

DEVICE real-valued function

LATCHED real-valued function

WINDOW program instruction

WINDOW real-valued function

SETDEVICE program instruction

Syntax

SETDEVICE (type, unit, error**, command)** p1, p2, ...

Function

Initialize a device or set device parameters. (The actual operation performed depends on the device referenced.)

Usage Considerations

The syntax contains optional parameters that apply only to specific device types and commands.

Parameters

type	Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced. The following types are currently available: 0 = Belt encoder 1 = (Not used) 2 = Force Processor Board (for Adept use only) 3 = Robot device (i.e., servo, for Adept use only) 4 = Vision 5 = 1394 bus (for Adept use only)
unit	Real value, variable, or expression (interpreted as an integer) that indicates the device unit number. The value must be in the range 0 to (max -1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real variable that receives a standard system error number that indicates if this instruction succeeded or failed. If this parameter is omitted, any device error stops program execution. If an error variable is specified, the program must explicitly check it to detect errors.
command	Real value, variable, or expression that specifies which device command or parameters are being set by this instruction. Some commands are standard and recognized by all devices. Other commands apply only to particular device types.
p1, p2, ...	Optional real values, variables, or expressions, the values of which are sent to the device as data for a command. The number of parameters specified and their meanings depend upon the particular device type being accessed.

Details

SETDEVICE is a general-purpose instruction for initializing external devices. It initializes the software and allows various parameters associated with the device to be set.

Two standard SETDEVICE commands are recognized by all devices:

command = 0 **Initialize device**
This command should be issued once before accessing the device with any other command. Normally, no additional parameters are required, but some device types may permit them.

command = 1 **Reset device**
This command resets the device. Normally no additional parameters are required, but some device types may permit them.

See the supplementary documentation for specific devices for details and examples.

For information on using the SETDEVICE instruction to access external encoders, see the section External Encoder Device in the *eV+ Language User's Guide*.

Related Keywords

DEVICE program instruction

DEVICE real-valued function

DEVICES program instruction

SHIFT transformation function

Syntax

SHIFT (transformation BY x_shift, y_shift, z_shift**)**

Function

Return a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.

Parameters

transformation	Transformation expression that is to be shifted.
x_shift y_shift z_shift	Optional real-valued expressions that are added to the respective position components of the transformation parameter.

Details

The value returned is equal to the value of the input transformation parameter value except that the three shift parameter values are added to the X, Y, and Z position components. If any shift parameter is omitted, its value is assumed to be zero.

Example

If the transformation x has the value:

```
(200, 150, 100, 10, 20, 30)
```

then executing the instruction:

```
SET y = SHIFT(x BY 5,-5,10)
```

results in the transformation y receiving the value:

```
(205, 145, 110, 10, 20, 30)
```

Related Keywords

SCALE transformation function

TRANS transformation function

SIG real-valued function

Syntax

SIG (signal_num, ..., signal_num)

Function

Returns the logical AND of the states of the indicated digital signals.

Parameter

signal_num	Real-valued expression that evaluates to a digital I/O or internal signal number. A negative value indicates negative logic for that signal.

Details

Returns a TRUE (-1) or FALSE (0) value obtained by performing a logical AND of the states of all the indicated digital signals. That is, SIG will return TRUE if all the specified signal states are TRUE. Otherwise, SIG will return FALSE.

The magnitude of each **signal_num** parameter determines which digital or internal signal is to be considered. Signals 1 - 8 and 33 - 512 are digital outputs. Signals 1001 - 1012 and 1033 - 1512 are digital inputs. Signals 2001 to 2512 are internal (software) inputs or outputs. Only digital signals that are actually installed can be used. You can use the IO monitor command (or the SIG.INS function) to check your current digital I/O configuration. Signals 3001 and 3002 refer to the robot selected by the current task. Signal 3001 is the state of the hand-close solenoid. Signal 3002 is the state of the hand-open solenoid.

If the sign of a **signal_num** parameter is positive, the signal is interpreted as being TRUE if it has a high value. If the sign of a **signal_num** parameter is negative, the signal is interpreted as being TRUE if it has a low value.

> **NOTE:** SIG(0) returns a value of TRUE.

Example

Assume that the following digital I/O signals are installed and have the indicated values.

- Input signal 1001 is On
- Input signal 1004 is Off
- Input signal 33 is Off

The following SIG function references return the indicated values:

```
SIG(1001)           ;Returns   -1.0 (TRUE)
SIG(1004)           ;Returns    0.0 (FALSE)
SIG(-1004)          ;Returns   -1.0 (TRUE)
SIG(1001,1004)      ;Returns    0.0 (FALSE)
SIG(1001,-1004)     ;Returns   -1.0 (TRUE)
```

Related Keywords

BITS monitor command

BITS program instruction

BITS real-valued function

IO monitor command

RESET monitor command

RUNSIG program instruction

SIGNAL monitor command

SIGNAL program instruction

SIG.INS real-valued function

Syntax

SIG.INS (signal_num)

Function

Return an indication of whether a digital I/O signal is installed in the system, or whether a software signal is available in the system.

Parameter

signal_num	Real-valued expression that defines the number of the digital I/O or software signal to check. (The absolute value is used, so negative signal numbers are allowed.)

Details

This function returns TRUE (-1) if the specified digital I/O or software signal is available for use by the system. Otherwise, FALSE (0.0) is returned. The function always returns TRUE if signal_number is zero.

This function can be used to make sure the digital I/O signals are installed as expected by the application program.

Example

The following program segment checks whether digital I/O signal #12 is installed as an input signal (referenced as signal #1012). A message is displayed on the system terminal if the signal is not configured correctly:

```
in.sig = 1012
IF NOT SIG.INS(in.sig) THEN
    TYPE "Digital I/O signal ", in.sig, "is not installed"
END
```

Related Keywords

BITS monitor command

BITS program instruction

BITS real-valued function

IO monitor command

RESET monitor command

RUNSIG program instruction

SIG.INS real-valued function

SIGNAL monitor command

SIGNAL program instruction

SIGN real-valued function

Syntax

SIGN (value)

Function

Return the value 1, with the sign of the value parameter.

Parameter

value Real-valued expression.

Details

This function returns -1.0 if the value of the parameter is less than zero. If the parameter value is greater than or equal to zero, +1.0 is returned.

Example

```
SIGN(0)           ;Returns 1.0
SIGN(0.123)       ;Returns 1.0
SIGN(-5.462)      ;Returns -1.0
SIGN(1.3125E+2)   ;Returns 1.0
```

SIGNAL program instruction

Syntax

SIGNAL signal_num, ..., signal_num

Function

Turn on or off external digital output signals or internal software signals.

Parameter

signal_num	Real-valued expression that evaluates to a digital output or internal signal number. A positive value indicates turn on; a negative value indicates turn off. (SIGNAL ignores parameters with a zero value.)

Details

The magnitude of a **signal_num** parameter determines which digital or internal signal is to be considered. Only digital output signals (numbered from 1 to 8 and 33 to 512) and internal (software) signals (numbered from 2001 to 2512) can be specified. Only digital signals that are actually installed and configured as outputs can be used. To check your current digital I/O configuration, use the IO monitor command. Signals 3001 and 3002 refer to the robot selected by the current task. Signal 3001 is the state of the hand-close solenoid. Signal 3002 is the state of the hand-open solenoid.

If the sign of the **signal_num** parameter is positive, the signal is turned on. If the sign of the **signal_num** parameter is negative, the signal is turned off.

> **NOTE:** All eV+ digital output instructions do not wait for a 16 millisecond eV+ cycle, they are turned on immediately. However, digital inputs are checked every 16 milliseconds by the eV+ operating system. Allowing the possibility to turn on and off a signal before the system can read the output.

Examples

Turn off the external output signal specified by the value of the variable reset (assuming the value of reset is positive), and turn on external output signal #4:

```
SIGNAL -reset, 4
```

Turn external output signal #1 off, external output signal #4 on, and internal software signal #2010 on:

```
SIGNAL -1, 4, 2010
```

Related Keywords

BITS monitor command

BITS program instruction

BITS real-valued function

IO monitor command

NOOVERLAP program instruction

OVERLAP program instruction

RESET monitor command

RUNSIG program instruction

SIG real-valued function

SIG.INS real-valued function

SIGNAL monitor command

SIN real-valued function

Syntax

SIN (value)

Function

Return the trigonometric sine of a given angle.

Usage Considerations

The angle parameter must be measured in degrees.

The parameter is interpreted as modulo 360 degrees, but excessively large values may cause a loss of accuracy in the returned value.

Parameter

value	Real-valued expression that defines the angular value to be considered.

Details

Returns the trigonometric sine of the argument, which is assumed to have units of degrees. The resulting value is always in the range of -1.0 to +1.0, inclusive.

Examples

```
SIN(0.123)       ;Returns 2.146753E-03
SIN(-5.462)      ;Returns -0.09518556
SIN(30)          ;Returns .5
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command displays real values to full precision.

SINGLE program instruction

Syntax

SINGLE ALWAYS

Function

Limit rotations of the robot wrist joint to the range -180 degrees to +180 degrees.

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.

MULTIPLE ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The SINGLE instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the SINGLE instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes SINGLE as the default condition. That is, if ALWAYS is specified, SINGLE remains in effect continuously until disabled by a MULTIPLE instruction. If ALWAYS is not specified, the SINGLE instruction applies only to the next robot motion.

Details

When moving to a transformation-specified location, the robot normally moves the wrist joint the minimum distance necessary to achieve the required orientation. In some cases, this action can move the wrist close to a limit stop so that a subsequent straight-line motion hits the stop.

Specifying SINGLE causes subsequent motion(s) to force the wrist back to near the center of its range, so that straight-line motions will not fail in this way.

SINGLE is commonly specified during an APPRO to pick up an object whose position and orientation were unknown at robot programming time. Once the object is acquired, the wrist motion can be kept to a minimum.

The SINGLE setting is ignored if NOOVERLAP is in effect.

Related Keywords

CONFIG real-valued function

MULTIPLE program instruction

NOOVERLAP program instruction

OVERLAP program instruction

SELECT program instruction

SELECT real-valued function

SOLVE.ANGLES program instruction

Syntax

SOLVE.ANGLES o.jts[*o.idx***], o.flags, error = trans, i.jts[***i.idx***],i.flags**

Function

Compute the robot joint positions (for the current robot) that are equivalent to a specified transformation.

Usage Considerations

Since the computation performed by this instruction is a function of the geometry of the robot (link dimensions, number of axes, tool offsets, base offsets), robots with different geometric parameters yields different results. In fact, since robots of the same general type may differ slightly in their dimensions, this instruction may return slightly different results when executed on two different robot systems of the same type.

The SOLVE.ANGLES instruction returns information for the robot selected by the task executing the instruction.

If the eV+ system is not configured to control a robot, executing this instruction does not generate an error because of the absence of a robot. However, the information returned may not be meaningful.

Parameters

o.jts Real-valued array in which the computed joint angles are returned. The first specified element of the array contains the position for joint #1, the second element contains the value for joint #2, etc. For rotating joints, the joint positions are in degrees. For translational joints, the joint positions are in millimeters.

If a computed joint position is outside the working range for the joint, the limit stop closest to the initial joint position (as indicated by i.jts[ ]) is returned.

o.idx Optional real value, variable, or expression (interpreted as an integer) that identifies the array element to receive the position for joint #1. If no index is specified, array element zero contains the position for joint #1.

o.flags Real variable that receives a bit-flag value that indicates the configuration of the robot corresponding to the computed joint positions. The bit flags are interpreted as follows:

Bit Flag	Description
Bit 1 (LSB) RIGHTY (mask value = 1)	If this bit is set, the position has the robot in a right-arm configuration. Otherwise, the position has the robot in a left-arm configuration.
Bit 2 BELOW (mask value = 2)	If this bit is set, the position has the robot configured with the elbow below the line from the shoulder to the wrist. Otherwise, the robot elbow is above the shoulder-wrist line. (This bit is always 0 when a SCARA robot is in use.)
Bit 3 FLIP (mask value = 4)	If this bit is set, the position has the robot configured with the pitch axis of the wrist set to a negative angle. Otherwise, the pitch angle of the robot wrist has a positive value. (This bit is always 0 when the robot does not have a three-axis wrist, which is the case for a SCARA robot.)

error Real variable that receives a bit-flag value that indicates whether any joint positions were computed to be outside of their working range, or whether the XYZ position of the destination was outside the working envelope of the robot. The bit flags are interpreted as follows:

<table>
<tr><th>Bit Flag</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Bits 1 - 12
Joint/Motor out of range</td><td colspan="2">If set, the computed value for the joint or motor was found to be outside of its limit stops:</td></tr>
<tr><td>Bit</td><td>Joint/ Motor #
Mask Value</td></tr>
<tr><td>1</td><td>1
^H1</td></tr>
<tr><td>2</td><td>2
^H2</td></tr>
</table>

Bit Flag	Description		
	3	3	^H4
	4	4	^H8
	5	5	^H10
	6	6	^H20
	7	7	^H40
	8	8	^H80
	9	9	^H100
	10	10	^H200
	11	11	^H400
	12	12	^H800
Bit 13 Collision (mask value = ^H1000)	When this bit is turned on, a collision has been detected.		
Bit 14 Too close (mask value = ^H2000)	The XYZ position of the destination cannot be reached because it was too close to the column of the robot.		
Bit 15 Too far (mask value = ^H4000)	The XYZ position of the destination cannot be reached because it was too far away from the robot.		
Bit 16 Joint vs. motor (mask value = ^H8000)	If set, a motor is limiting. Otherwise, a joint is limiting.		

trans Transformation variable, function, or compound transformation that defines the robot location of interest.

i.jts Real array that contains the joint positions representing the starting

location for the robot. These values are referenced: (1) for multiple-turn joints to minimize joint rotations, and (2) when a computed joint position is out of range to determine which limit stop to return.

The first specified element of the array must contain the position for joint #1. The second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to be in degrees. For translational joints, the joint positions are assumed to be in millimeters.

i.idx Optional real value, variable, or expression (interpreted as an integer) that identifies the array element that contains the position value for joint #1. If no index is specified, element zero must contain the position for joint #1.

i.flags Real value, variable, or expression whose value is interpreted as bit flags that indicate: (1) the initial configuration of the robot, (2) any changes in configuration that are to be made, and (3) special operating modes. The bit flags are interpreted as follows:

Bit Flag	Description
Bit 1: (LSB) RIGHTY (mask value = 1)	If this bit is set, the robot is assumed initially to be in a right-arm configuration. Otherwise, the robot is assumed to be in a left-arm configuration.
Bit 2: BELOW (mask value = ^H2)	If this bit is set, the robot is assumed initially to have its elbow below the line from the shoulder to the wrist. Otherwise, the robot is assumed to have its elbow above that line. (This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.)
Bit 3: FLIP (mask value = ^H4)	If this bit is set, the robot is assumed initially to have the pitch axis of the wrist set to a negative value. Otherwise, the pitch angle is assumed to be set to a positive value. This bit is ignored if the robot does not have a three-axis wrist.

Bit Flag	Description
Bit 9: Change RIGHTY/LEFTY (mask value = ^H100)	If this bit is set, the instruction attempts to compute a set of joint positions corresponding to the RIGHTY/LEFTY configuration specified by bit 10.
Bit 10: Change to RIGHTY (mask value = ^H200)	When bit 9 is set and this bit is set, the instruction attempts to compute joint positions for a right-arm configuration. If bit 9 is set and this bit is 0, the instruction attempts to compute a set of joint positions for a left-arm configuration.
Bit 11: Change BELOW/ABOVE (mask value = ^H400)	If this bit is set, the instruction attempts to compute a set of joint positions corresponding to the BELOW/ABOVE configuration specified by bit 12. This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.
Bit 12: Change to BELOW (mask value = ^H800)	When bit 11 is set and this bit is set, the instruction attempts to compute joint positions for an elbow-down configuration. If bit 11 is set and this bit is 0, the instruction attempts to compute joint positions for an elbow-up configuration. This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.
Bit 13: Change FLIP/NOFLIP (mask value = ^H1000)	If this bit is set, the instruction attempts to compute a set of joint positions corresponding to the FLIP/NOFLIP configuration specified by bit 14. This bit is ignored if the robot does not have a three-axis

Bit Flag	Description
	wrist.
Bit 14: Change to FLIP (mask value = ^H2000)	When bit 13 is set and this bit is set, the instruction attempts to compute joint positions for a FLIP wrist configuration. If bit 13 is set and this bit is 0, the instruction attempts to compute joint positions for a NOFLIP wrist configuration. This bit is ignored if the robot does not have a three-axis wrist.
Bit 21: Avoid degeneracy (mask value = ^H100000)	When this bit is set, if the computed value of joint #2 is within 10 degrees of having the outer link straight out (that is, joint 2 between -10 and +10 degrees in value), an out-of-range error for joint 2 is signaled.
Bit 22: Single-turn joint 4 (mask value = ^H200000)	When this bit is set, the computed value of joint 4 is restricted to the range of -180 to +180 degrees.
Bit 23: Straight-line motion (mask value = ^H400000)	When this bit is set, the joint positions returned must correspond to the same configuration as those initially specified. That is, no change in robot configuration is allowed.

Details

This instruction computes the joint positions that are equivalent to a specified transformation value using the geometric data of the robot connected to the system. The specified transformation is interpreted to be the position and location of the end of the robot tool in the World coordinate system, taking into consideration the current TOOL transformation and BASE offsets.

Example

The instructions below do not perform any useful function but are intended to illustrate how the SOLVE.ANGLES instruction operates. After execution of these instructions, both the jts2 and jts arrays contain approximately the same values. Any differences in the values are due to computational round-off errors:

```
HERE #cpos
DECOMPOSE jts[] = #cpos
SOLVE.TRANS new.t, error = jts[]
SOLVE.ANGLES jts2[], flags, error = new.t, jts[], SOLVE.FLAGS(jts
[])
```

Related Keywords

DECOMPOSE program instruction

SELECT program instruction

SELECT real-valued function

SOLVE.FLAGS real-valued function

SOLVE.TRANS program instruction

SOLVE.FLAGS real-valued function

Syntax

SOLVE.FLAGS (joints[index**])**

Function

Return bit flags representing the robot configuration specified by an array of joint positions.

Usage Considerations

The SOLVE.FLAGS function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the SOLVE.FLAGS function causes an error.

Parameters

joints	Real array that contains the robot joint positions. The first specified element of the array must contain the position for joint #1, the second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to have units of degrees. For translational joints, the joint positions are assumed to have units of millimeters.
index	Optional real value, variable, or expression (interpreted as an integer) that identifies the array element that contains the position for joint #1. If no index is specified, element zero must contain the position for joint #1.

Details

This function returns bit flags that indicate the configuration of the robot (for example, RIGHTY or LEFTY) for a given set of joint positions. This function is useful for providing the configuration data required by the SOLVE.ANGLES program instruction.

The bits of the value returned by this function are interpreted as follows:

Bit 1 (LSB) RIGHTY (mask value = 1)

If this bit is set, the position has the robot in a right-arm configuration. Otherwise, the position is for a left-arm configuration.

Bit 2 BELOW (mask value = 2)

If this bit is set, the position has the robot configured with the elbow below the line from the shoulder to the wrist. Otherwise, the robot elbow is above the shoulder-wrist line. (This bit is always 0 when a SCARA robot is in use.)

Bit 3 FLIP (mask value = 4)

If this bit is set, the position has the robot configured with the pitch axis of the wrist set to a negative angle. Otherwise, the wrist pitch angle has a positive value. (This bit is always 0 when the robot does not have a three-axis wrist, as is the case for a four-axis SCARA robot.)

Related Keywords

ABOVE program instruction

BELOW program instruction

DECOMPOSE program instruction

LEFTY program instruction

RIGHTY program instruction

FLIP program instruction

NOFLIP program instruction

SELECT program instruction

SELECT real-valued function

SOLVE.ANGLES program instruction

SOLVE.TRANS program instruction

SOLVE.TRANS program instruction

Syntax

SOLVE.TRANS transform, error = joints[index**]**

Function

Compute the transformation equivalent to a given set of joint positions for the current robot.

Usage Considerations

Since the computation performed by this instruction is a function of the geometry of the robot (link dimensions, number of axes, tool offsets, base offsets), robots with different geometric parameters yield different results. In fact, since robots of the same general type may differ slightly in their dimensions, this instruction may return slightly different results when executed on two different robot systems of the same type.

The SOLVE.TRANS instruction refers to the robot selected by the task executing the instruction.

If the eV+ system is not configured to control a robot, executing the SOLVE.TRANS instruction does not generate an error because of the absence of a robot. However, the information returned may not be meaningful.

Parameters

transform	Transformation variable or transformation array element in which the result is stored.
error	Real variable that is set to a eV+ error code if a computational error occurred during processing of the instruction. This variable is set to 0 if no error occurs. (The only error that is currently reported is arithmetic overflow [-409], so this parameter can be considered as returning a TRUE or FALSE value.)
joints	Real-valued array that contains the joint positions that are to be converted to an equivalent transformation. The first specified element of the array must contain the position for joint #1, the second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to have units of degrees. For translational joints, the joint positions are assumed to have units of millimeters.
index	Optional integer value that identifies the array element that contains the position for joint #1. If no index is specified, element zero must contain the position for joint #1.

Details

This instruction converts a set of joint positions to an equivalent transformation value using the geometric data of the robot connected to the system. The computed transformation represents the position and orientation of the end of the tool in the World coordinate system taking into consideration the current TOOL transformation and BASE offsets.

Example

The series of instructions below computes the position and orientation to which the robot is moved if its current location is altered by rotating joint #1 by 10 degrees:

```
HERE #cpos
DECOMPOSE joints[1] = #cpos
joints[1] = joints[1]+10
SOLVE.TRANS new.trans, error = joints[1]
```

Related Keywords

DECOMPOSE program instruction

SELECT program instruction

SELECT real-valued function

SOLVE.ANGLES program instruction

SOLVE.FLAGS real-valued function

SPEED program instruction

Syntax

SPEED speed_factor, r_speed_factor units *ALWAYS*

Function

Set the nominal speed for subsequent robot motions.

Usage Considerations

SPEED 100,100 ALWAYS is assumed whenever program execution is started and when a new execution cycle begins.

Motion speed has different meanings for joint-interpolated motions and straight-line motions.

The speed of robot motions is determined by a *combination* of the program speed setting and the monitor speed setting.

The SPEED instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the SPEED instruction causes an error.

Parameters

speed_factor	Real value, variable, or expression whose value is used as a new speed factor. The value 100 is considered normal full speed, 50 is 1/2 of full speed, and so on. If IPS or MMPS is specified for units, the value is considered the linear tool tip speed.
r_speed_factor	Optional real value, variable, or expression whose value is used as a new straight-line motion rotational speed factor. The value 100 is considered normal full speed, 50 is 1/2 of full speed, etc.
units	Optional keyword—either IPS (for inches per second), MMPS (for millimeters per second), or MONITOR—that determines how to interpret the **speed_factor** parameter.
ALWAYS	Optional qualifier. If specified, the program **speed_factor** will be in effect until the next SPEED instruction changes program speed. Otherwise, it is in effect only for the next motion instruction (including APPROaches and DEPARTs).

Details

If the units parameter is omitted, this instruction determines the program speed—the nominal robot motion speed, assuming that the monitor speed factor is 100%.

If MONITOR is specified for units, the monitor speed is set. In this case, the parameter r_speed_factor is ignored and ALWAYS is assumed. The speed at which motions are actually performed is determined by combining the values specified in this instruction with the current program speed setting. Monitor speed changes take place immediately, including the remaining portion of a currently executing move.

If IPS or MMPS is specified in the units parameter, **speed_factor** is interpreted as the absolute tool-tip speed for *straight-line* motions. In this case, the **speed_factor** parameter has no direct meaning for joint-interpolated motions.

The effects of changing program speed and monitor speed differ slightly for continuous-path motions. As the robot moves through a series of points, the robot comes as close to the points as possible while maintaining the program speed and specified accelerations. As program speed increases, the robot makes coarser approximations to the actual point in order to maintain the program speed and accelerations.

When the monitor speed is increased, the path of the robot relative to the commanded destination points is not altered but the accelerations are increased. For applications where path following is important, the path can be defined with the monitor speed set to a low value, and then accurately replayed at a higher monitor speed.

Speed cannot be less than 0.000001 (1.0E-6).

When the Monitor speed is set, its value is limited to a maximum of 100%. No error is reported if a higher speed setting is specified.

When the program speed is set, its value is limited to a maximum that depends on the robot being controlled. No error is reported if a higher speed setting is specified. The maximum speed value for the current robot is returned by the real-valued function SPEED(8).

During straight-line motions, if a tool with a large offset is attached to the robot, the robot joint and flange speeds can be very large when rotations about the tool tip are made. The r_speed_factor parameter permits control of the maximum tool rotation speeds during straight-line motions.

If a rotational speed factor (r_speed_factor) is specified, it is interpreted as a percentage of maximum Cartesian rotation speed to be used during straight-line motions. If the r_speed_factor parameter is not specified, one of the following results occurs:

1. If the units parameter is also omitted, the rotational speed is set to the value of **speed_factor**.
2. If the units parameter is specified, the rotational speed is not changed.

When IPS or MMPS are specified, the **speed_factor** is converted internally to the corresponding nominal speed. If the SPEED real-valued function is then used to read the program speed, the value returned is a percentage speed factor and not an absolute speed setting.

Remember, the final robot speed is a combination of the monitor speed (SPEED monitor command), the program speed (SPEED instruction), and the acceleration or deceleration (ACCEL program instruction).

Examples

Set the program speed to 50% for the next motion (assuming the monitor speed is 100):

```
SPEED 50
```

Set the nominal tool tip speed to 20 inches per second (assuming the monitor speed is 100) for straight-line motions. Rotations about the tool tip is limited to 40% of maximum. The settings remains in effect until changed by another SPEED instruction.

```
SPEED 20, 40 IPS ALWAYS
```

Set the monitor speed to 50% of normal:

```
SPEED 50 MONITOR
```

Related Keywords

ACCEL program instruction

DURATION program instruction

IPS keyword

MMPS keyword

SCALE.ACCEL system switch

SELECT program instruction

SELECT real-valued function

SPEED monitor command

SPEED real-valued function

SPEED real-valued function

Syntax

SPEED (select)

Function

Return one of the system motion speed factors.

Usage Considerations

The SPEED function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the SPEED function does not generate an error because of the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select Real-valued expression whose value determines which speed factor should be returned (see below).

Details

This function returns the system motion speed factor corresponding to the select parameter value. The acceptable parameter values, and the corresponding speed values returned, are:

Select	Speed value returned
1	Monitor speed (set by SPEED monitor command)
2	Permanent program speed (set by a SPEED ... ALWAYS program instruction)
3	Temporary program speed for the last or current motion
4	Temporary program speed to be used for the next motion
5	Permanent program rotation speed
6	Temporary program rotation speed for the last or current straight-line motion

Select	Speed value returned
7	Temporary program rotation speed to be used for the next straight-line motion
8	The maximum allowable setting for program speed

Note that the value returned should be interpreted as a percentage of normal speed, even if the program speed was set by a SPEED program instruction that specified a speed setting. (See the SPEED program instruction.)

Example

The following program segment makes one motion at 1/2 of the permanent program speed:

```
new.speed = SPEED(2)/2  ;Compute 1/2 the permanent speed
SPEED new.speed         ;Move at the new speed next time
MOVE pick.up            ;Perform the actual motion
```

Note that the following instruction sequence is equivalent:

```
SPEED SPEED(2)/2        ;Reduce speed for the next motion
MOVE pick.up            ;Perform the actual motion
```

Related Keywords

ACCEL real-valued function

DURATION real-valued function

SELECT program instruction

SELECT real-valued function

SPEED monitor command

SPEED program instruction

SQR real-valued function

Syntax

SQR (value)

Function

Return the square of the parameter.

Parameter

value Real-valued expression whose value is to be squared.

Details

This is a convenience function that computes the square of a value. That is, the result is equal to (value * value).

Examples

```
SQR(0.123)              ;Returns 0.015129
SQR(4)                  ;Returns 16
SQR(-5.462)             ;Returns 29.83344
SQR(1.3125E+2)          ;Returns 17226.56
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command displays real values to full precision.

SQRT real-valued function

Syntax

SQRT (value)

Function

Return the square root of the parameter.

Parameter

value	Real-valued expression defining the value whose square root is to be computed.

Details

Returns the square root of the argument if the argument is greater than zero. An error results if the argument is less than zero.

The square root of a number is defined to be the number that, when multiplied by itself, yields the original number.

Examples

```
SQRT(0.123)             ;Returns 0.3507136
SQRT(4)                 ;Returns 2.0
SQRT(-5.462)            ;Returns *Negative square root*
SQRT(1.3125E+2)         ;Returns 11.45644
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LIST and LISTR monitor command displays real values to full precision.

STATE real-valued function

Syntax

STATE (select)

Function

Return a value that provides information about the robot system state.

Usage Considerations

The STATE function returns information for the robot selected by the task executing the function.

Parameter

select Real value, variable, or expression (interpreted as an integer) that selects the category of state information returned. The information categories are listed below. For details on a category, click the category name to view its description.

Robot State (select=1)

System Settings

Current manual control mode (select=3)

Hardware status (select=4)

Settings of controller switches on the SmartController EX (select=5)

Trajectory generator execution rate (select=13)

Number of robot selected by the pendant (select=8)

Robot Motion Information

Current or previous robot motion (select=2)

Status of Real-time path-modification facility (alter mode) for current motion= (select=6 and select= 7)

Time until completion of robot motion (select=9)

Percentage of current motion completed (select=10)

Acceleration profile (select=11)

Number of motion being executed by the selected robot (select=15)

Determine whether current or previous motion is optimized (select=19)

Planned execution time of current motion (select=20-27)

Details

When **select** = 1, the function value returns information about the overall robot state as follows:

Value	Interpretation (when select = 1)
0	Resetting system after robot power has been turned off.
1	A fatal error has occurred and robot power cannot be turned on.
2	Waiting for user to turn on robot power.
3	Robot power was just turned on; initialization is occurring.
4	Indicates that Manual control mode is active (see Select=3).
5	A CALIBRATE command or instruction is executing.
6	Not used.
7	Robot is under program control.
8	Robot power is on; robot is not calibrated and cannot be moved.
10	Front panel power light is blinking slowly (1 Hz), waiting to be pressed. Robot power will be turned on when the blinking button is pressed.
11	Front panel power light is blinking rapidly (4 Hz). Robot power will be turned on when: the COMP/PWR button on the pendant is pressed. NOTE: Pressing the front panel power light when it is blinking rapidly will cause robot power to be turned off fully. The normal process will be required to turn power back on.

When **select** = 2, the function value returns information about the current or previous robot motion as follows. These modes can change only when the robot is under program control-that is, when STATE(1) = 7.

Value	Interpretation (when select = 2)
0	No motion instructions executed yet.
1	Normal trajectory evaluation is in progress (including normal acceleration, deceleration and segment transitions).
2	Motion stopped at a planned location.[1]
3	Position error is being nulled at unplanned final location.
4	Motion stopped at an unplanned location due to a belt window violation.[2]
5	Decelerating due to a triggered REACTI or BRAKE instruction.
6	Stopped due to a triggered REACTI or BRAKE instruction.[2]
7	Decelerating due to a hardware error, panic button, or ESTOP instruction.
8	Stopped due to a hardware error, panic button, or ESTOP instruction.[2]
9	Decelerating due to a stop-on-force condition.
10	Stopped due to a stop-on-force condition.
11	Nulling at completion of a SPIN motion.
12	Stopped after completion of a SPIN instruction.
[1]A RETRY command has no effect. [2]A RETRY command completes the previous motion.	

When **select** = 3, the function value returns information about the current manual control mode as follows:

Value	Interpretation (select = 3)
0	Manual mode without selection.
1	Free-joint mode.
2	Individual joint control.
3	World coordinates control.
4	Tool coordinates control.
5	Computer control enabled.

When **select** = 4, the function value returns information about thehardware status to be read by programs. Interpret the value as a set of bit flags, each of which indicates a corresponding condition.

Value	Interpretation when bit set (select = 4)
^H1	Not used.
^H2	Not used.
^H4	ESTOP circuit is open[1].
^H8	HIGH POWER button is pushed.[2]
^H10	ESTOP channel 1 is open.
^H20	ESTOP channel 2 is open.
^H40	Front Panel keyswitch is in manual state.
^H80	Not used.
^H100-^HF00	ESTOP source: 0x0 = No ESTOP or pending (waiting for 120 ms settling period) 0x1 = ESTOP from loss of ESTOP source

Value	Interpretation when bit set (select = 4)
	0x2 = ESTOP from Front Panel 0x3 = ESTOP from pendant 0x4 = ESTOP from User Gate 0x5 = ESTOP from Line Input 0x6 = ESTOP from Muted Safety Gate (auto mode only) 0x7 = ESTOP from AUTO to Manual change 0x8 = ESTOP from Manual to AUTO change 0x9 - 0xE = Reserved for future use 0xF = Unresolved ESTOP source
^H1000	3-position Enable switch is closed (reported only in manual mode).
[1] If eV+ is detecting a hardware ESTOP condition Bit 4 is non-zero. [2]The HIGH POWER button will not function if its light bulb is burned out or missing.	

When **select** = 5, the function value indicates the settings of the switch on the Front Panel. For more information, refer to the *Adept SmartController User's Guide*.

Value	Interpretation (select = 5)
1	Automatic mode.
2	Manual mode.

When **select** = 6, the function returns an indication of whether or not the real-time path-modification facility (alter mode) is enabled. If zero is returned, alter mode is disabled for the current motion. If a nonzero value is returned, alter is enabled, and the low byte of this value contains bits that correspond to the mode specified in the ALTON instruction that initiated the path modification.

When **select** = 7, the function returns an indication of whether or not the real-time path-modification facility (alter mode) is enabled for the next planned motion. If zero is returned, alter mode is disabled for the next motion. If a nonzero value is returned, alter is enabled, and the low byte of this value contains bits that correspond to the mode specified in the ALTON instruction that initiated the path modification. (This option is available only if your Adept system is equipped with the eV+ Advanced Trajectory Control License.)

When **select** = 8, the number of the robot selected by the manual control pendant is returned.

When **select** = 9, the function returns the time (in seconds) left until completion of the current motion. Zero indicates that no motion is in progress. For continuous-path motions, the value of STATE(9) decreases during each motion until the transition to the next motion, and then the value suddenly changes to the time left in the next motion. That is, STATE(9) does not reach 0 before it is reset to reflect the next motion.

When **select** = 10, the function returns the percentage of the current motion that has completed. The value 100 indicates that no motion is in progress. For continuous-path motions, the value of STATE(10) increases during each motion until the transition to the next motion, and then the value suddenly changes to close to 0 to reflect the start of the next motion. That is, STATE(10) does not reach 100 before it is reset to reflect the next motion.

When **select** = 11, the function returns detailed information on which portion of the acceleration profile is currently being generated for the selected robot.

Value	Interpretation (select = 11)
0	Idle, not evaluating trajectory
1	Ramping up acceleration for new segment
2	Constant acceleration section
3	Ramping down acceleration
4	Constant velocity section
5	Ramping up acceleration during the transition section between motions
6	Constant acceleration during the transition section between motions
7	Ramping down acceleration during the transition section between motions
8	Ramping up deceleration
9	Constant deceleration
10	Ramping down deceleration
11	Nulling final errors

When **select** = 12, the function returns a flag that is set to nonzero when an ALTER program instruction is executed for the currently selected robot, and cleared after the trajectory generator processes the posted ALTER data. This flag can be used to coordinate the execution of ALTER instructions with the processing of the data by the trajectory generator.

When **select** = 13, the function returns the trajectory generator execution rate in Hertz. That is, if the trajectory generator is executed every 16ms, this function returns the value 62.5.

When **select** = 15, the function returns the number of the motion that is being executed by the selected robot. This number is zeroed when a program that is attached to the robot first begins executing. The counter is reset to 1 at the start of each EXECUTE cycle. The value is incremented each time the trajectory generator begins evaluating a new motion (or transitions to a new continuous-path motion). The value of STATE(15) ranges from 0 to ^HFFFF. After reaching ^HFFFF, the value rolls back to 0.

The following instructions affect the value of STATE(15):

> ALIGN, APPRO, APPROS, DEPART, DEPARTS, DRIVE, JMOVE, MOVE, MOVES, MOVEF, MOVESF, MOVET, MOVEST, READY

Instructions that affect one or more subsequent motions (e.g., ACCEL, AMOVE, ABOVE, BELOW, DURATION, FLIP, LEFTY, NOFLIP, MULTIPLE, RIGHTY, SINGLE, SPEED, SPIN, TOOL, UNIDIRECT, ...) do not affect the value of STATE(15), because those instructions do not actually initiate a motion.

NOTE: Functions do not affect the value of STATE(15), because a function does not cause a motion. Location-valued functions (e.g., DEST, FRAME, INVERSE, SCALE, SHIFT, TRANS, etc.) simply compute location values.

When **select** = 19, the function returns a flag that indicates if the current motion (or the previously executed motion if the robot is stopped) is an optimized move (i.e., MOVEF or MOVESF). The possible values returned by this function are:

Value	Interpretation (select = 19)
0	Not a MOVEF/MOVESF motion.
1	Depart segment of MOVEF/MOVESF
2	Horizontal segment of MOVEF/MOVESF
3	Final segment of MOVEF/MOVESF

When **select** is 20 through 27, the function returns detailed information on the planned execution time of the current motion (or the previously executed motion if the robot is

stopped). Note, unlike STATE(9) that returns the remaining motion execution time corrected for the monitor speed setting, the values returned by STATE(20) through STATE(27) are the planned values and are not affected by the setting of the monitor speed value. The values returned by these functions (in units of seconds) are:

select	Information returned
20	Acceleration ramp up time
21	Constant acceleration time
22	Acceleration ramp down time
23	Constant velocity time
24	Deceleration ramp up time
25	Constant deceleration time
26	Deceleration ramp down time
27	Total motion time (sum of STATE(20) through STATE(26))

When **select** = 30, the function returns the state of the Front Panel power light. The possible values returned by this function are:

Value	Interpretation (select = 30)
0	Light is off.
1	Light is on.
2	Light is blinking at 4 Hz.
3	Light is blinking at 1 Hz.

Example

The following example shows how the STATE function can be used to determine whether or not a REACTI was triggered during a robot motion:

```
REACTI 1001         ;Setup the reaction
MOVES final         ;Start the robot motion
BREAK               ;Wait for the motion to complete
CASE STATE(2) OF    ;Decide what happened
  VALUE 2:
```

```
    TYPE "Motion completed normally"
  VALUE 6:
    TYPE "Motion stopped by REACTI"
  VALUE 8:
    TYPE "Motion stopped by panic button"
END
```

Related Keywords

SELECT program instruction

SELECT real-valued function

STATUS monitor command

STATUS real-valued function

STATUS real-valued function

Syntax

STATUS (program_name)

Function

Return status information for an application program.

Usage Considerations

The use of STATUS with a null program name to determine the task number is obsolete. The TASK real-valued function is more efficient and should be used instead.

Parameter

program_name	String constant, variable, or expression that specifies the name of the application program of interest. Letters in the name can be uppercase or lowercase. The string can be empty ( ) in order not to specify a program name (see below), but the parameter cannot be omitted.

Details

This function returns information about the execution status of the specified program.

If no program name is specified (that is, the parameter string is empty [ ]), the task number of the program containing the function call is returned. This allows a program to determine which system task it is executing as. (Tasks and task numbers are described in the section Scheduling of Program Execution Tasks in the *eV+ Language User's Guide*.)

If a program name is specified as the function parameter, the status of that program is returned as follows:

Value returned	Program status
-1	Not executing.
-2	Not defined.
-3	Interlocked because of write.

Value returned	Program status
-4	Not executable.
-5	Interlocked because of read.

A program is considered write-interlocked when it is being copied, deleted, renamed, or edited in read-write mode. A program is considered read-interlocked when it is executing (by one or more tasks) or is being edited in read-only mode.

A program is considered not executable when it contains a structure error or a bad line.

NOTE: If a program is being executed by multiple tasks, the STATUS function returns -5. There is no way to use the STATUS function to determine when the program ceases to be executed by one of those tasks. The STATUS function does not return -1 until all the tasks stop executing the program.

The function returns a not defined status if an invalid program name is specified (for example, if the name does not start with a letter).

Example

The following program segment demonstrates how the STATUS function can be used to decide whether or not to initiate execution of an application program:

```
IF STATUS("pc.main") == -1 THEN
    EXECUTE 1 pc.main
END
```

NOTE: The STATUS function does not return -1 if the program is being executed by any program task. Thus, this example may not be appropriate for some situations. (See the example shown for the EXECUTE instruction for another technique for initiating execution of another program task.)

Related Keywords

DEFINED real-valued function

STATE real-valued function

STATUS monitor command

TESTP monitor command

STOP program instruction

Syntax

STOP

Function

Terminate execution of the current program cycle.

Usage Considerations

STOP does not halt program execution if there are more program cycles to execute.

The PROCEED command cannot be used to resume program execution after a STOP instruction causes the program to halt.

If program execution is halted by a STOP instruction, FCLOSE and/or DETACH are forced on all attached I/O devices.

Details

Counts one more program cycle as complete and one less remaining. If the result is that no more cycles are remaining, program execution halts.

If more cycles are remaining, the internal robot motion parameters are reinitialized, and program execution continues with the first step of the main program (even if the STOP occurred within a subroutine or reaction program).

Terminates execution of the current program unless more program loops (see the EXECUTE command and instruction) are to be completed, in which case execution of the program continues at its first step. Thus, the STOP instruction is used to mark the end of a program execution pass. Note that the HALT instruction has a different effect-it cancels all remaining cycles.

A RETURN instruction in a main program has the same effect as a STOP instruction. A main program is one that is invoked by an EXECUTE command or instruction, or a PRIME or XSTEP command, whereas a subroutine is a program that is invoked by a CALL, CALLP, or CALLS instruction (or a reaction) within another program.

Related Keywords

ABORT monitor command

ABORT program instruction

EXECUTE program instruction

HALT program instruction

PAUSE program instruction

RETURN program instruction

STRDIF real-valued function

Syntax

STRDIF ($a, $b)

Function

Compare two strings byte by byte for the purpose of sorting. This function always compares bytes exactly. It ignores the setting of the UPPER system switch.

Parameters

$a	A string constant, variable, or expression that contains the bytes to be compared with those in **$b**.
$b	A string constant, variable, or expression that contains the bytes to be compared with those in **$a**.

Details

This function compares strings byte by byte, using the unsigned byte values without any case conversion. That is, the function ignores the setting of the UPPER system switch. The two strings can have different lengths. The returned values and their meanings are as follows:

Returned value	Interpretation
-1	**$a** is less than **$b**.
0	**$a** is exactly the same as **$b**.
1	**$a** is greater than **$b**.

Note that the value is FALSE (0) if the strings are the same.

Example

Sort two names in alphabetical order:

```
$name[0] = "Michael"
$name[1] = "MARK"
CASE STRDIF($name[0],$name[1]) OF
  VALUE -1, 0:
```

```
    $list[0] = $name[0]
    $list[1] = $name[1]
  VALUE 1:
    $list[0] = $name[1]
    $list[1] = $name[0]
END
TYPE "Names in alphabetic order:  ", $list[0], " ", $list[1]
```

Related Keyword

UPPER system switch

SWITCH program instruction

Syntax

SWITCH switch_name = value

SWITCH switch_name[index**] = value**

Function

Enable or disable a system switch based on a value.

Usage Considerations

If the specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), **all** the elements of the switch array are set according to the value given.

Parameters

switch_name	Name of the switch whose setting is to be modified. The switch name can be abbreviated to the minimum length that identifies it uniquely.
index	For switches that can be qualified by an index, this is an optional real value, variable, or expression that specifies the specific switch element of interest (see above).
value	Real value, variable, or expression that determines whether the switch is to be enabled or disabled. The switch is enabled if the value is TRUE (nonzero). The switch is disabled if the value is FALSE (zero).

Details

Sets the given system switch to the setting implied by the value on the right of the equal sign.

The switch name can be abbreviated to the minimum length that identifies it uniquely.

For details on switch names, see the section Switches in the *eV+ Language User's Guide*.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the *AdeptVision Reference Guide*.

Example

The following program statements show how the SWITCH real-valued function and instruction can be used to save the setting of a system switch, and later restore it, respectively:

```
old.upper = SWITCH(UPPER)       ;Save the current setting
          .
          .                     ;Instructions that may change the
          .                     ;setting of the UPPER switch.

SWITCH UPPER = old.upper        ;Restore the initial setting
```

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH real-valued function

SWITCH real-valued function

Syntax

SWITCH (switch_name)

SWITCH (switch_name[index**])**

Function

Return an indication of the setting of a system switch.

Parameters

switch_name	Name of the system switch of interest (see below).
index	For switches that can be qualified by an index, this is a (required) real value, variable, or expression that specifies the specific switch element of interest.

Details

This function returns FALSE (0.0) if the specified switch is disabled. Otherwise, TRUE (-1) is returned.

The switch name can be abbreviated to the minimum length that identifies it uniquely.

For details on switch names, see the section Switches in the *eV+ Language User's Guide*.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the *AdeptVision Reference Guide*.

Example

This program segment checks whether the DRY.RUN switch is enabled. If it is, a message is displayed on the system terminal:

```
IF SWITCH(DRY.RUN) THEN
    TYPE "DRY RUN mode is enabled"
END
```

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

$SYMBOL string function

Syntax

$SYMBOL (pointer)

Function

Determine the user symbol that is referenced by a pointer previously obtained with the SYMBOL.PTR real-valued function.

Usage Considerations

The pointer value must have been obtained with the SYMBOL.PTR real-valued function.

Parameter

pointer Real variable that identifies the symbol to be referenced.

Details

This function can be used to determine the user symbol (that is, program or variable name) that is pointed to by a pointer previously determined with the SYMBOL.PTR real-valued function.

A null string is returned if the pointer value is zero or invalid, or if the symbol has been deleted since the pointer was defined.

Example

After the SYMBOL.PTR function has been used to set the values of elements of the array **my.pgm.ptr[ ]** (for example, see the dictionary page for the CALLP instruction), the following instruction can be used to display the program name that is referenced by one of the pointers:

```
TYPE "Program", index, " is ", $SYMBOL(my.pgm.ptr[index])
```

Related Keyword

CALLP program instruction

SYMBOL.PTR real-valued function

SYMBOL.PTR real-valued function

Syntax

SYMBOL.PTR (string, type**)**

Function

Determine the value of a pointer to a user symbol in eV+ memory.

Usage Considerations

The value returned by the function is meaningful only to the CALLP instruction and the $SYMBOL string function.

Parameters

string	String constant, variable, or expression that defines the symbol to be referenced.
type	Optional real value, variable, or expression that specifies the type of symbol to be referenced. Currently the only value supported is zero, which specifies that the string parameter defines a program name. The value zero applies if the parameter is omitted.

Details

The SYMBOL.PTR function can be used to obtain a pointer to a user symbol (that is, a program or variable name) in eV+ memory. Such a pointer can then be used elsewhere in the program by the CALLP instruction and the $SYMBOL function. Refer to the descriptions of those keywords for more information.

The function returns the value zero if the specified symbol is not defined.

Example

Refer to the dictionary page for the CALLP instruction.

Related Keywords

CALLP program instruction

$SYMBOL string function

$SYS.INFO string function

Syntax

$string **= $SYS.INFO(mode,**$stg**)**

Function

This function provides general system information.

Input Parameters

mode — Determines the type of information returned based on the following mode values:

Value	Description
0	Returns the full file specification (device>subdirectory_path\filename.extension) for the disk file from which the eV+ program module specified by *$stg* was loaded. If the module is undefined or was not loaded from a file, an empty string value is returned.

If the mode is invalid, $SYS.INFO returns an empty string rather than generating an execution error.

$stg — An optional string valued expression.

Details

Returns the information requested by the mode and $string input parameters. If the mode is invalid, $SYS.INFO returns an empty string rather than generating an execution error.

TAS real-valued function

Syntax

TAS (variable, new_value)

Function

Return the current value of a real-valued variable and assign it a new value. The two actions are done indivisibly so that no other program task can modify the variable at the same time.

Usage Considerations

The eV+ system does not enforce any protection scheme for global variables that are shared by multiple program tasks. It is the programmer's responsibility to keep track of the usage of such global variables. The TAS real-valued function (or the similar CAS function) can be used to implement logical interlocks on access to shared variables.

This function can also be used to work around a restriction on the simultaneous access of global arrays by multiple program tasks -- program execution can fail if two or more tasks attempt to increase the size of an array at the same time. For a detailed description of this, see the "Global Array Access Restriction" section of the information about Arrays, in the *eV+ Language User's Guide*.

Parameters

variable	Name of the real-valued variable to be tested and assigned the new value given. (If the variable is not defined when the function is executed, the function returns the value 0.)
new_value	Real value, variable, or expression that defines the new value to be assigned to the specified variable.

Details

Because the different program tasks execute simultaneously, time-sharing the system processor, it is possible for any task to be interrupted by another in the middle of performing some computation or storing data into variables. When data is shared by two or more tasks, the programs must implement an interlock scheme to prevent the data from being accessed when it is only partially updated.

The TAS function can be used to allow multiple eV+ tasks to modify shared data structures. That is, the function provides a way for a task to lock out others while the locking task modifies the data structures. Note that without the TAS function, a much more complicated polling scheme would be needed to administer the control variable (i.e., to prevent more than one program from setting the control variable simultaneously).

As an example of the use of shared variables, consider this eV+ program that increments and decrements a global variable:

```
.PROGRAM tas_test()
        AUTO i
        FOR i = 1 TO 1E+06
            counter = counter+1
            counter = counter-1
        END
.END
```

If the variable *counter* starts at 0, and the program *tas_test* is run **simultaneously** in tasks 1 and 2, one might expect that *counter* will have the value 0 after execution completes. In fact, it usually will not have that value! Since the two tasks are modifying the same variable at the same time, the value gets corrupted.

That can be fixed by modifying the program to employ an interlock as follows:

```
.PROGRAM tas_test()
        AUTO i
        FOR i = 1 TO 1E+06
            ; Wait for access to the shared variable to be
            ;        unlocked, and set the lock for our access.
            WHILE TAS(locked,TRUE) DO
                WAIT
            END
            ; Access the shared variable.
            counter = counter+1
            counter = counter-1
            ; Release access to the shared variable.
            locked = FALSE
        END
.END
```

Now, when the program is executed simultaneously in two (or more) tasks. The value of the variable *counter* will always end up the same as before the program starts. That's because each task blocks the other task(s) while accessing the shared variable.

(The global variable *locked* does not need to be initialized before the program is executed, because the TAS function returns FALSE if the variable is not defined. Thus, the lock implicitly starts out being off.)

> **NOTES:** The lock should be released as soon as possible, because the other task could be waiting for it to be released.
>
> Take care to make sure the lock is *always* released after it gets applied. Otherwise the other task could be blocked forever, and the *current* task would also be blocked the next time it tries to acquire the lock.

Example

The following example shows the key aspects of using the TAS real-valued function to ensure exclusive access by an application program to data that is also used by another program task. (The same instruction sequence must be used in any other application program that wants to access the data.)

The real-variable *data.locked* has the value FALSE when the data is not interlocked, and the value TRUE when the data is interlocked. This variable is set to TRUE with the TAS function, so that we can detect if the other program task has already set it to TRUE. Since TAS tests and sets the value indivisibly, there is no chance of both programs setting *data.locked* to TRUE simultaneously without the conflict being detected.

Use of the "semaphore" variable *data.locked* involves the three steps shown below.

```
; Step1: Look for the lock variable to have the "unlocked" setting (FALSE),
;        and simultaneously apply the "lock" setting (TRUE).  This loop will
cycle
;        continuously until another task sets the lock variable to the
"unlocked"
;        setting (FALSE), at which time this task asserts the lock for
itself.

        WHILE TAS(data.locked,TRUE) DO   ;Wait for the lock to be released
            WAIT                         ;Sleep this task until the next
cycle
        END
; Step 2: Perform desired operations accessing the shared data ...
        ...
; Step 3: Release the lock on the shared data structure.
        data.locked = FALSE
```

The WHILE loop causes program execution to be blocked until the variable *data.locked* is found to have the value FALSE. Thus, the program is blocked if the other program has locked the semaphore variable in order to access the shared data. Note that the TAS function will set the variable *data.locked* to TRUE each time the function is executed, but that will have no effect if the variable already has that value.

Once the program gains exclusive access to the shared data, it can safely access the data.

The last instruction releases the data for access by the application executing as the other program task.

Related Keyword

CAS real-valued function

TASK real-valued function

Syntax

TASK (select, task_num**)**

Function

Return information about a program execution task.

Parameters

select	Optional real-valued expression that has a value of 0, 1, or 2 and selects the category of task information returned (see below). The value 0 is assumed if the parameter is omitted.
task_num	Optional integer value that specifies which system program task is to be accessed (see below).

Details

This function returns various information about the system program execution tasks. (See the *eV+ Language User's Guide* for an explanation of execution tasks.)

The select parameter determines the type of information that is returned as follows:

select = 0 **Task number:** The function returns the number of the task executing the current program.

select = 1 **Task run state:** Returns the run state for the task specified by the task_num parameter. The value returned should be interpreted as follows:

Value	Interpretation
-1	Invalid task number.
0	Idle.
1	Stopped due to program completion.
2	Stopped due to program execution error (for example, undefined value).

Value	Interpretation
3	Stopped due to ABORT, panic button pressed, robot error, or watchpoint.
4	Executing.
5	Stopped due to PAUSE or breakpoint
7	Stopped due to single-step execution

select = 2 **Task status bits:** Returns an integer value that should be interpreted as a set of bit flags that indicate the following information about the task specified by the task_num parameter:

Bit #	Bit mask	Indication if bit is set
1	1	Debugger is accessing task
2	2	Task has robot attached

Examples

Display the task number the program is running in:

```
TYPE "This program is running as task number :", TASK()
```

The following program segment demonstrates how the TASK function can be used to decide whether to initiate execution of a program (named pc.job.2) with task #2:

```
IF TASK(1,2) <> 4 THEN                  ;If task #2 not executing
    IF STATUS("pc.job.2") == -1 THEN    ;and if program is okay
        EXECUTE 2 pc.job.2()            ;start it up
    ELSE                                ;But if program not okay
        TYPE /B, "Can't start task #2"  ;output error message
    END
END
```

Related Keywords

ERROR real-valued function

STATE real-valued function

STATUS monitor command

STATUS real-valued function

TIME program instruction

Syntax

TIME time_string

Function

Set the date and time.

Parameter

time_string String expression whose value specifies the date and time to be set. The value of the string must have one of these formats (see below):

```
dd-mmm-yy hh:mm:ss          dd-mmm-yyyy hh:mm:ss
dd-mmm-yy hh:mm             dd-mmm-yyyy hh:mm
```

Details

The system clock is set equal to the value of the string expression.

The system clock is maintained automatically and should be changed only when its setting is incorrect (e.g., the controller is moved to a different time zone).

The system clock is used in the following situations:

- The date and time are displayed when the eV+ system is booted from disk.
- Whenever a new disk file is created, the date and time are recorded with the file name. (The FDIRECTORY command displays the dates and times for files.)
- The date and time are appended to the message indicating that an application program has terminated execution.
- The date and time are displayed by the TIME monitor command.
- The date and time are available to an application program by use of the $TIME() and $TIME4() string functions.

The individual elements of the date and time specification are defined as follows:

Element	Description
dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)

Element	Description
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
yyyy	The year (1980 to 2079)
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

Example

```
TIME "23-JUN-99 16:10:25"
```

Related Keywords

TIME monitor command

TIME real-valued function

$TIME string function

$TIME4 string function

TIME real-valued function

Syntax

TIME (string**, select)**

Function

Return an integer value representing either the date or the time specified in the given string parameter.

Parameters

string — Optional string variable, constant, or expression that specifies the date and time in the format described below. (See below for details.)

select — Real value, variable, or expression (interpreted as an integer) that selects the value to be returned. An error results if **select** is not one of the following:

select	Returned	Defined ss
1	date	(year-1980)*512 + month*32 + day
2	time	hour*2048 + minute*32 + second/2
3	seconds	time past the minute

Details

This function can be used to encode the date and time into compact (unsigned 16-bit) integer formats. After the integer date and time values are obtained, they can be arithmetically compared to other date and time values to determine before and after conditions.

> **NOTE:** You should not try to manipulate the encoded integer values to perform date or time arithmetic. For example, you should not attempt to add days to an encoded date value.

If the string parameter is supplied, both the date and the time must be specified in the string. The value of the string must have one of the following formats:

```
dd-mmm-yy hh:mm:ss      dd-mmm-yyyy hh:mm:ss
dd-mmm-yy hh:mm         dd-mmm-yyyy hh:mm
```

(The function returns the value -1 if the input string does not have an acceptable format [see the example below].)

The individual date and time elements are defined as follows:

Element	Description
dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
yyyy	The year (1980 to 2079)
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

If the string parameter is not supplied and the **select** parameter is 1, the current date of the system clock is returned. In addition, the current time of the system clock is stored in the (internal) administrative data for the program task. If the string parameter is not supplied and the **select** parameter is 2 or 3, the selected time value is returned for the system-clock time *previously* saved.

Example

The following program segment shows how the TIME real-valued function can be used to make sure you enter a valid date and time after a prompt:

```
PROMPT "Enter the date and time (dd-mmm-yy hh:mm:ss): ", $time
WHILE (TIME($time,1) == -1) DO          ;Make sure it's valid
    TYPE /B, "    Cannot interpret date/time."
    PROMPT "Try again (dd-mmm-yy hh:mm:ss): ", $time
END
TIME $time                              ;Set system time
```

Related Keywords

TIME monitor command

TIME program instruction

$TIME string function

$TIME4 string function

$TIME string function

Syntax

$TIME (date, time**)**

Function

Return a string value containing either the current system date and time or the specified date and time.

Parameters

date Optional integer value representing the year, month, and day (see below). The value is interpreted as follows (month ranges from 1 to 12):

```
date = (year-1980)*512 + month*32 + day
```

time Optional integer value representing the hour, minutes, and seconds past midnight (see below). The value is interpreted as follows (hour ranges from 0 to 23):

```
time = hour*2048 + minute*32 + second/2
```

NOTE: This function always returns a string containing both the date and the time. That can result in an erroneous date string if the date parameter is omitted when the time parameter is specified.

Details

If both the date and time parameters are omitted, this function returns the current system date and time in the format described below. (An empty string is returned if the system clock has not been initialized.)

If the date and time parameters are specified, their values are converted to an ASCII string in the format described below, and the string is returned. This operation is used to decode the output values generated by the TIME real-valued function.

The date and time are output in the format dd-mmm-yy hh:mm:ss, in which the individual elements are defined as follows:

Element	Description
dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR,

Element	Description
	MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59)

NOTE: The $TIME() function converts passed arguments (instead of the system time) when either the date or the time parameter is supplied. However, the function always tries to generate a string representation of both parameters. You get the date 01-Jan-80 if you do not provide a date value. The time substring is 00:00:00 if you do not specify a time value. The following expressions can be used to return only the date and the time, respectively:

```
$date = $MID($TIME(date,),1,9)

$time = $MID($TIME(,time),11,8)
```

Related Keywords

TIME monitor command

TIME program instruction

TIME real-valued function

$TIME4 string function

$TIME4 string function

Syntax

$TIME4 (date, time**)**

Function

Return a string value containing either the current system four-digit date and time or the specified four-digit date and time.

Parameters

date	Optional integer value representing the year, month, and day (see below). The value is interpreted as follows (month ranges from 1 to 12): `date = (year-1980)*512 + month*32 + day`
time	Optional integer value representing the hour, minutes, and seconds past midnight (see below). The value is interpreted as follows (hour ranges from 0 to 23): `time = hour*2048 + minute*32 + second/2`

NOTE: This function always returns a string containing both the date and the time. That can result in an erroneous date string if the date parameter is omitted when the time parameter is specified.

Details

If both the date and time parameters are omitted, this function returns the current system date and time in the format described below. (An empty string is returned if the system clock has not been initialized.)

If the date and time parameters are specified, their values are converted to an ASCII string in the format described below, and the string is returned. This operation is used to decode the output values generated by the TIME real-valued function.

The date and time are output in the format dd-mmm-yyyy hh:mm:ss, in which the individual elements are defined as follows:

Element	Description
dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR,

Element	Description
	MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yyyy	The year (1980 to 2079)
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59)

NOTE: The $TIME4() function converts passed arguments (instead of the system time) when either the date or the time parameter is supplied. However, the function always tries to generate a string representation of both parameters. You get the date 01-Jan-80 if you do not provide a date value. The time substring is 00:00:00 if you do not specify a time value. The following expressions can be used to return only the date and the time, respectively:

```
$date = $MID($TIME4(date,),1,11)

$time = $MID($TIME4(,time),13,8)
```

Related Keywords

TIME monitor command

TIME program instruction

TIME real-valued function

$TIME string function

TIMER program instruction

Syntax

TIMER timer_number = time_value

Function

Set the specified system timer to the given time value.

Usage Considerations

Times measured by eV+ are precise only to within 1 millisecond (0.001 seconds); shorter times cannot be measured.

Timers with numbers <= 0 are read-only and cannot be set with this instruction.

Parameters

timer_number	Real-valued expression interpreted as the (integer) number of the timer to be set. The value must range from 1 to 15.
time_value	Real-valued expression interpreted as the time, in seconds, to which the timer is set. This parameter may specify fractions of a second and may be negative.

Details

When used as described in the examples below, the timers can be used to measure an interval of 596 hours from when they were set by the TIMER instruction. Timers have a resolution of one millisecond and a maximum count of > 2.E+009.

Use the TIMER real-valued function to read the instantaneous value of a system timer.

Example

The following examples show two ways to wait for a certain amount of time, using the TIMER instruction and real-valued function. Each example first sets the timer, and then waits until the timer value has changed by the delay period:

```
TIMER 1 = 0                 ;Set timer to zero
WAIT TIMER(1) > delay       ;Wait until timer > delay
TIMER 1 = -delay            ;Set timer to -delay
WAIT TIMER(1) > 0           ;Wait until timer > zero
```

Related Keyword

TIMER real-valued function

TIMER real-valued function

Syntax

TIMER (timer_number)

Function

Return the current time value of the specified system timer.

Usage Considerations

The accuracy and resolution of the timers vary according to which timer is selected. DOUBLE variables should be used to achieve maximum resolution. See the *Details* section below.

Parameter

timer_number Real value, variable, or expression (interpreted as an integer) that specifies the number of the timer to be read. The value must be in the range -4 to 15.

Value	Description
1-15	Timers with a resolution of one millisecond and a maximum count of > 2.E+009. They can be used to measure an interval of up to 596 hours from when they were set by the TIMER instruction.
0	Returns the number of seconds since the eV+ system was started, with a resolution of 1 millisecond and a maximum count of about 2.E+009. It is valid only during the first 596 hours of system operation and should generally not be used.
-1	Returns the low 24 bits of the time since the eV+ system was started, in counts of 16 milliseconds. It can be used to compute time intervals of up to 74 hours. For additional information, see the *Details* section below.
-2	Returns the low 24 bits of the time since the eV+ system was started, in counts of 1

Value	Description
	millisecond. It can be used to compute time intervals of up to 4.6 hours. For additional information, see the *Details* section below.

-3	Returns the time in seconds since eV+ was started as a 52-bit double-precision value. This timer has a resolution of 1 millisecond and a maximum count of > 4.E+015. It can be used to compute intervals of > 100,000 years. It is used like timers -1 and -2 except that DOUBLE variables are required and no BAND operation or scale factors are used.
	`AUTO DOUBLE start_time, interval`
	`start_time = TIMER(-3)`
	`...`
	`interval = TIMER(-3)-start_time`

-4	Returns the double-precision time of the current robot-position or belt-encoder latch for this task. The timer resolution is 1 microsecond. This time will only be valid for 128 seconds.

Details

Timers -1 and -2

If you don't want to use timers 1 through 15, or need more than 15 timers, Timers -1 and -2 may be used as follows:

```
AUTO DOUBLE start_time, interval, scale
scale = 62.5                    ;Set scale = 1000 for TIMER(-2)
start_time = TIMER(-1)
...
interval = ((TIMER(-1)-start_time) BAND ^HFFFFFF)/scale
```

Note that **timer -3** provides a better method for computing such intervals provided that a DOUBLE value can be used.

The type of eV+ variable used in time computations affects the maximum interval that can be computed with full resolution:

- Standard REAL variables have only 24 bits of resolution, which limits the time interval to 16,777,216 (2^{24}) counts. This limit corresponds to about 4.6 hours for millisecond

timers and 74 hours for 16-millisecond timers.

- DOUBLE REAL variables have 52 bits of precision, which stores the full resolution of the various timers. This is the default type used when none is explicitly specified.

Example

The following example shows how the TIMER instruction and real-valued function can be used to time the execution of a subroutine:

```
TIMER 1 = 0            ;Set timer to zero
CALL test.routine()    ;Call the subroutine
TYPE "Elapsed time =", TIMER(1)," seconds"
```

Related Keyword

TIMER program instruction

TOOL program instruction

Syntax

TOOLtransformation_value

Function

Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.

Usage Considerations

The TOOL instruction causes a BREAK in continuous-path motion.

The TOOL instruction can be executed by any program task as long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the eV+ system is not configured to control a robot, executing the TOOL instruction causes an error.

The word "tool" cannot be used as a program name or variable name.

Parameter

transformation_value	Optional transformation variable or function, or compound transformation expression, that is the new tool transformation. If the transformation value is omitted, the tool is set to NULL.

Details

Causes a BREAK in the robot continuous-path motion and sets the value of the tool transformation equal to the transformation value given.

Refer to the TOOL monitor command for a complete description of the effect of this instruction. (For information on how to define a tool transformation, see the section Tool Transformations in the *eV+ Language User's Guide*.)

Related Keywords

SELECT program instruction

SELECT real-valued function

TOOL monitor command

TOOL transformation function

TOOL transformation function

Syntax

TOOL

Function

Return the value of the transformation specified in the last TOOL command or instruction.

Usage Considerations

The command LISTL TOOL can be used to display the current tool setting.

The TOOL function returns information for the robot selected by the task executing the function.

If the eV+ system is not configured to control a robot, use of the TOOL function does not generate an error because of the absence of a robot. However, the information returned by the function may not be meaningful.

The name "tool" cannot be used as a program name or variable name.

Examples

Display the value of the current TOOL transformation from the system prompt:

```
LISTL TOOL
```

Save the value of the current TOOL:

```
SET save.tool = TOOL
```

Related Keywords

SELECT program instruction

SELECT real-valued function

TOOL monitor command

TOOL transformation function

TPS real-valued function

Syntax

TPS

Function

Return the number of ticks of the system clock that occur per second (Ticks Per Second).

Usage Considerations

The name "tps" cannot be used as a program name or variable name.

Example

The following example shows how an event can be tested each system clock tick, with a time-out of 5 seconds, using the TPS function and the WAIT instruction.

```
FOR ticks = 1 TO 5*TPS      ;Loop 5*ticks/sec times
    IF SIG(1001) THEN
        TYPE "Signal ON"
        HALT
    END
    WAIT                    ;Wait until next clock tick
END
TYPE "Time-out while waiting for signal 1001"
```

TRANS transformation function

Syntax

TRANS (X_value, Y_value, Z_value, y_value, p_value, r_value**)**

Function

Return a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.

Parameters

X_value Y_value Z_value	Optional expressions for the X, Y, and Z displacement components, respectively.
y_value p_value r_value	Optional expressions for the yaw, pitch, and roll orientation components, respectively.

NOTE: If any parameter is omitted, its value is taken to be zero.

Details

The input parameter values are used to compute a transformation value that can be assigned to a location variable or used in a compound transformation or motion request.

Examples

If **r** s the radius of a circle and **angle** is the angle of rotation about the circle, then the transformation:

```
TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
```

yields points on that circle.

If **frame** is a transformation defining the position of the center of the circle and the plane in which it lies, the following program segment moves the robot tool point around the circle in steps of 1 degree.

```
FOR angle = 0 TO 360-1
    MOVE frame:TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
END
```

Related Keywords

DECOMPOSE program instruction

DX real-valued function

DY real-valued function

DZ real-valued function

#PPOINT precision-point function

SET program instruction

SHIFT transformation function

TRANSB transformation function

$TRANSB string function

Syntax

$TRANSB (transformation, double_precision**)**

Function

Return a 48-byte or 96-byte string containing the binary representation of a transformation value.

Parameter

transformation	Transformation variable or function (or compound transformation) that defines the value to be converted to a string value.
double_precision	Optional real-valued expression that specifies wether the transformation is returned in single-precision (48 bytes) or double-precision (96 bytes). If double_precision is omitted or has a value of 0, the transformation is returned in single precision in a 48-byte string, otherwise the transformation is returned in double precision in a 96-byte string. This parameter is used to offer full precision while preserving backward compatibility.

Details

This function converts the given transformation value to the binary representation of its twelve (internal) components. The twelve values defining the transformation are the components of a 3-by-4 transformation matrix, stored by column. Each of the twelve 32-bit values is packed as four successive 8-bit characters in a string, resulting in a total of 48 or 96 characters. (The IEEE single-precision or double-precision standard floating-point format is used for the conversion. See the description of the FLTB or DBLB real-valued function for details of the IEEE floating-point format.)

The main use of this function is to convert a transformation value to its binary representation in an output record of a data file.

Related Keywords

$FLTB string function

FLTB real-valued function

TRANSB transformation function

TRANSB transformation function

Syntax

TRANSB (string, first_char, double_precision**)**

Function

Return a transformation value represented by a 48-byte or 96-byte string.

Parameters

string	String expression that contains the 48 or 96 bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the 48 bytes in the string. If first_char is omitted or has a value of 0 or 1, the first 48 or 96 bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second through 49th or 97th bytes are extracted. An error is generated if first_char specifies 48 or 96 bytes that are beyond the end of the input string.
double_precision	Optional real-valued expression that specifies wether the transformation is represented by a single-precision (48 bytes) or double-precision (96 bytes) string. If double_precision is omitted or has a value of 0, the transformation is represented by a single precision in a 48-byte string, otherwise the transformation is represented by a double precision in a 96-byte string. This parameter is used to offer a full precision mode while maintaining backward compatibility.

Details

48 or 96 sequential bytes of the given string are interpreted as being a set of twelve single-precision (32-bit) or double-precision (64-bit) floating-point numbers in the IEEE standard format. (See the description of the FLTB or DBLB function for details of the floating-point format.) The twelve values are interpreted as the components of a 3-by-4 transformation matrix, stored by column.

The main use of this function is to convert the binary representation of a transformation value from an input data record to values that can be used internally by eV+.

Related Keywords

FLTB real-valued function

TRANS transformation function

$TRANSB string function

TRUE real-valued function

Syntax

... TRUE

Function

Return the value used by eV+ to represent a logical true result.

Usage Considerations

The word "true" cannot be used as a program name or variable name.

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is –1.

Example

The following program loop executes continuously until the subroutine process returns a TRUE value for the real variable error:

```
DO
    CALL process(error)
UNTIL error == TRUE
```

The program loop below will execute indefinitely:

```
WHILE TRUE DO
    CALL move.part()
END
```

Related Keywords

FALSE real-valued function

ON real-valued function

$TRUNCATE string function

Syntax

$TRUNCATE (string)

Function

Return all characters in the input string until an ASCII NUL (or the end of the string) is encountered.

Parameter

string	String variable, constant, or expression that specifies the string to be truncated.

Details

This function is similar to performing a $DECODE operation with an ASCII NUL (^H00) specified as the break character. $TRUNCATE differs from such a $DECODE operation in two ways:

- The input can be a string expression.
- The input string is not modified.

Because of its simplicity, the $TRUNCATE function executes much faster than the $DECODE function.

Example

The instruction below sets the value of the string variable $substring equal to abcdef. (Obviously, this is an artificial situation, since one would never want to perform a $TRUNCATE operation when the result is apparent from the input. However, it is presented to illustrate that this function can scan an arbitrary string expression and return the first substring delimited by a NUL.)

```
$substring = $TRUNCATE("abcdef"+$CHR(0)+"ghijk")
```

Related Keyword

$DECODE string function

TYPE program instruction

Syntax

TYPE output_specification, ..., output_specification

Function

Display the information described by the output specifications on the system terminal. A blank line is output if no argument is provided.

Usage Considerations

No output is generated if the MESSAGES system switch is disabled.

Program execution normally waits for the output to be completed before continuing. There is an output specification described below that can be used to prevent waiting if it is undesirable for execution to be delayed.

The output from a single TYPE instruction cannot exceed 512 characters. (The /S format control specifier described below can be used to output longer messages.)

Parameter

An output_specification can consist of any of the following components (in any order) separated by commas:

1. A string expression.
2. A real-valued expression, which is evaluated to determine a value to be displayed.
3. Format-control information, which determines the format of the output message.

Details

The following format-control specifiers can be used to control the way in which numeric values are displayed. These settings remain in effect for the remainder of the instruction, unless another specifier is used to change their effect.

For all these display modes, if a value is too large to be displayed in the specified field width, the field is filled with asterisk characters (*).

/D Use the default format, which displays values to full precision with a single leading space. (Scientific notation is used for values greater than or equal to 1,000,000.)

> **NOTE:** The following format specifications accept a zero as the field width (n). That causes the actual field size to vary to fit the value, and causes all leading spaces to be suppressed. That is useful when a value is displayed within a line of text or at the end of a line.

/En.m	Output values in scientific notation (for example, -1.234E+02) in fields n spaces wide, with m digits the fractional parts. If n is not zero, it must be large enough to include space for a minus sign (if the displayed value is negative), one digit to the left of the decimal point, a decimal point (if m is not zero), m digits, and four or five characters for the exponent.
/Fn.m	Output values in fixed-point notation (for example, -123.4) in fields n spaces wide, with m digits in the fractional parts.
/Gn.m	Output values in F format with m digits in the fractional parts if the values are larger than 0.01 and will fit in fields n spaces wide. Otherwise /En.m format is used.
/Hn	Output values as hexadecimal integers in fields n spaces wide.
/In	Output values as decimal integers in fields n spaces wide.
/On	Output values as octal integers in fields n spaces wide.

The following specifiers can be used to control the appearance of the output.

/Cn	Output the characters carriage return (CR) and line feed (LF) n times. This will result in n blank lines if the control specifier is at the beginning or end of an output specification; otherwise, n-1 blank lines will result.
/S	Do not output a carriage return (CR) or line feed (LF) after displaying the current line.
/Un	Move the cursor up n lines. This will work correctly only if the TERMINAL parameter is correctly set for the terminal being used.
/Xn	Output n spaces.

The following specifiers can be used to perform control functions.

/B	Beep the terminal (nongraphics-based systems only).
/N	Initiate output without having program execution wait for its completion. A second output request will force program execution to wait for the first output if it has not yet completed.

Example

Assume that the real variable i has the value 5 and that array element point[5] has the value 12.666666. Then, the instruction

```
TYPE /B, "Point", i, " = " /F5.2, point[i]
```

sounds a beep at the system terminal (/B) and display the message

```
Point 5 = 12.67
```

If point[5] has the value 1000, the instruction displays

```
Point 5 = *****
```

because the value (1000.00) is too large to be displayed in the specified format (/F5.2). (The instruction can display any value for point[5] if the format specification were /F0.5.)

Related Keywords

$ENCODE string function

MESSAGES system switch

PROMPT program instruction

WRITE program instruction

$UNPACK string function

Syntax

$UNPACK (string_array[index**], first_char, num_chars)**

Function

Return a substring from an array of 128-character string variables.

Parameters

string_array	String array variable from which the substring is to be extracted. It is assumed that each string within the array is defined and is 128 characters long.
index	Optional integer value(s) that identifies the first array element to be considered. The **first_char** value is interpreted relative to the element specified by this index. If no index is specified, element zero is assumed.
first_char	Real-valued expression that specifies the position of the first character of the substring within the string array. A value of 1 corresponds to the first character of the specified string array element. This value must be greater than zero. The value of **first_char** can be greater than 128. In that case the array element accessed follows the element specified in the function call. For example, a value of 130 corresponds to the second character in the array element following that specified by index.
num_chars	Real-valued expression that specifies the number of characters to be returned by the function. This value can range from 0 to 128.

Details

This function extracts a substring from an array of strings. Substrings are permitted to overlap two string array elements. For example, a 10-character substring whose first character is the 127th character in element [3] is composed of the last two characters in element [3] followed by the first eight characters of element [4].

In order to efficiently access the string array, this function assumes that all of the array elements are defined and are 128 characters long. For multidimensional arrays, only the

right-most array index is incremented to locate the substring. Thus, for example, element [2,3] is followed by element [2,4].

Example

The instruction below sets the value of the string variable $substring equal to a substring extracted from the string array $list[]. The substring is specified as starting in element $list [3]. However, because the first character is to be number 130, the 11-character substring actually consists of the second through 12th characters of $list[4]:

```
$substring = $UNPACK($list[3], 130, 11)
```

Related Keywords

$MID string function

PACK program instruction

UNTIL program instruction

Syntax

UNTIL expression

Function

Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is nonzero.

Usage Considerations

UNTIL must be used in conjunction with a DO control structure. See the description of the DO instruction for details.

Parameter

expression	Real-valued expression, constant, or relation that is interpreted as either TRUE (nonzero) or FALSE (zero).

Details

If the expression in the UNTIL statement is zero, program execution continues with the statement following the matching DO statement. If the expression is nonzero, program execution continues with the statement following the UNTIL statement.

Example

The following example is a loop that continues to prompt you to enter a number until you enter one that is greater than or equal to zero:

```
DO
    PROMPT "Enter a positive number: ", number
UNTIL number >= 0
```

Related Keyword

DO program instruction

EXIT program instruction

NEXT program instruction

WHILE program instruction

UPPER system switch

Syntax

... UPPER

Function

Control whether or not the case of each character is ignored when string comparisons are performed.

Usage Considerations

The switch value is shared globally by all program tasks. If you change the value in one task, it affects comparisons in all other tasks. Therefore, do not change this switch during normal program execution.

Details

When this switch is enabled and two strings are compared using the operators <, <=, ==, <>, >=, or >, all lowercase characters are treated as though they were uppercase characters. That is, when UPPER is enabled, both of the following comparisons yields a TRUE value:

```
"a" == "A"      and     "A" == "A"
```

When UPPER is disabled, the case of characters is considered during string comparisons. Then, for example, the comparison on the left above results in a FALSE value, while the comparison on the right yields a TRUE value.

By default, UPPER is enabled, so that string comparisons are performed without considering the case of the characters.

The STRDIF real-valued function always compares strings considering their case. You can leave UPPER enabled always and then use STRDIF in situations where case is important.

Related Keywords

DISABLE monitor command

DISABLE program instruction

ENABLE monitor command

ENABLE program instruction

SWITCH monitor command

SWITCH program instruction

SWITCH real-valued function

STRDIF real-valued function

VAL real-valued function

Syntax

VAL (string)

Function

Return the real value represented by the characters in the input string.

Usage Considerations

The input string can be a number in scientific notation.

The input string can contain leading number base indicators (^H, for example).

The input string can contain a + or - sign before the numeric part of the string, but after any optional base indicator.

Any character that cannot be interpreted as part of a number or as a base indicator marks the end of the characters that are converted.

Parameter

string String constant, variable, or expression.

Examples

```
VAL("123 Elm Street")   ;Returns the real value 123
VAL("1.2E-2")           ;Returns the real value 0.012
VAL("^HFF")             ;Returns the real value 255
```

Related Keywords

ASC real-valued function

$ENCODE string function

FLTB real-valued function

INTB real-valued function

LNGB real-valued function

VALUE program instruction

Syntax

VALUE expression_list:

Function

Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.

Usage Considerations

VALUE must be part of a CASE control structure. See the description of the CASE instruction for details.

Parameter

expression_list	List of real values or expressions separated by commas.

Related Keywords

ANY program instruction

CASE program instruction

WAIT program instruction

Syntax

WAIT condition

Function

Put the program into a wait loop for one trajectory cycle. If a condition is specified, wait until the condition is TRUE.

Usage Considerations

- A WAIT with no condition specified is useful in programs that need to perform an operation only once each trajectory cycle. For more information, see Details and Example 1.
- To wait for a specific time period, use the WAIT.EVENT program instruction rather than the WAIT program instruction.
- During execution, use the PROCEED monitor command to cancel a WAIT instruction in an application program.

Parameter

condition	Optional real value, variable, or expression that is tested for a TRUE (nonzero) or FALSE (zero) value.

Details

If no condition is supplied with the WAIT instruction, program execution is suspended until the next trajectory cycle. Trajectory cycles occur at 16, 8, 4 or 2 millisecond intervals, depending on the system configuration.

If you need to guarantee at least a trajectory cycle delay (for example, while manipulating signals monitored by REACT or REACTI), you should execute *two* consecutive WAIT instructions (with no arguments).

If a condition is specified, WAIT will suspend program execution until the condition exists. For example, the state of one or more external signals can be used as the condition for continuation.

Example

Stop program execution while external input signal #1001 is on and #1003 is off. Poll once eacheV+ trajectory cycle:

```
WHILE SIG(1000, -1003) DO
```

```
    WAIT
END
```

Related Keywords

RELEASE program instruction

WAIT.EVENT program instruction

WAIT.START monitor command

WAIT.EVENT program instruction

Syntax

WAIT.EVENT mask, timeout

Function

Suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed.

Usage Considerations

If a WAIT.EVENT instruction in an application program has execution suspended, the WAIT.EVENT can be canceled with the PROCEED monitor command.

Parameters

mask	Optional real value, variable, or expression that specifies the events for which to wait. The value is interpreted as a sequence of bit flags, as detailed below. (All the bits are assumed to be clear if no mask value is specified.) Bit 1 (LSB) Wait for I/O (mask value = 1) If this bit is set, the desired event is the completion of any input/output operation by the current task.
timeout	Optional real value, variable, or expression that specifies the number of seconds to wait. No time-out processing is performed if the parameter is omitted, or the value is negative or zero (see below for more details).

Details

This program instruction is used to suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed in the timeout clock. The program waits efficiently.

When the program resumes execution after a WAIT.EVENT instruction, the GET.EVENT function can be used to verify that the desired event has actually occurred. This is the only way to distinguish between the occurrence of an event and a time-out (if one was specified).

If the mask parameter has the value zero (or is omitted), this instruction becomes a very efficient way to suspend program execution for the time period specified by the timeout parameter.

If the timeout parameter is omitted (or has a negative or zero value), this instruction suspends program execution indefinitely until the specified event occurs.

If both mask and timeout are zero or omitted, this instruction does nothing.

WAIT.EVENT 1 waits for an event to be signaled for a task. Events are signaled by either a SET.EVENT program instruction, or by a pending no-wait I/O instruction when the I/O operation is completed.

In general, there is no way to tell why the event was set. It may have been set by an I/O operation, a SET.EVENT program instruction, or an internal system process (such as a triggered REACT condition). For this reason, it is necessary to test for the desired condition after executing the WAIT.EVENT. For I/O, repeat the no-wait I/O operation or use the IOSTAT () function. For SET.EVENT issued by other tasks, define and check a global variable.

To avoid race conditions where the event is set or cleared between testing and waiting, use the following loop in the waiting task (the statement order is critical).

1. CLEAR.EVENT
2. Issue no-wait I/O if appropriate.
3. Check I/O status or check global variable.
4. Exit loop if operation complete.
5. WAIT.EVENT 1
6. GOTO step 1

If using SET.EVENT to signal another task, use the following sequence (the statement order is critical).

1. Set the global variable.
2. SET.EVENT for the appropriate task.

Examples

Suspend program execution for 5.5 seconds:

```
WAIT.EVENT , 5.5
```

Suspend program execution until the completion of any system input/output, or until another program task sets events using the SET.EVENT instruction:

```
WAIT.EVENT 1
```

Suspend program execution for five seconds, until the completion of any system input/output, or until another program task uses the SET.EVENT instruction to set events. (The current program should use the GET.EVENT function to decide whether an event has occurred or five seconds has elapsed.)

```
WAIT.EVENT 1, 5
```

Related Keywords

CLEAR.EVENT program instruction

GET.EVENT real-valued function

SET.EVENT program instruction

WHILE program instruction

Syntax

WHILE condition DO

Function

Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.

Usage Considerations

Every WHILE statement must be part of a complete WHILE ... DO ... END structure.

Parameter

condition	Real-valued expression that is evaluated and tested for a TRUE (nonzero) or FALSE (zero) value.

Details

This structure provides another means for executing a group of instructions until a control condition is satisfied (compare it with the DO structure). The complete syntax for the WHILE structure is

```
WHILE condition DO
    group_of_steps
END
```

Processing of the WHILE structure can be described as follows:

1. Evaluate the **condition**. If the result is FALSE, proceed to item 4.
2. Execute the group_of_steps.
3. Return to item 1.
4. Continue program execution at the first instruction after the END step.

Unlike the DO structure described elsewhere, the group of instructions within the WHILE structure may not be executed at all. That is, if the condition has a FALSE value when the WHILE is first executed, then the group of instructions are not executed at all.

When this structure is used, it is assumed that some action occurs within the group of enclosed instructions that will change the result of the logical expression from TRUE to FALSE when the structure should be exited.

Example

The following example uses a WHILE structure to monitor a combination of input signals to determine when a sequence of motions should be stopped. In this example, if the signal from either part feeder becomes zero (assumed to indicate the feeder is empty), then the repetitive motions of the robot stops and the program continues.

Note that if either feeder is empty when the WHILE structure is first encountered, then execution immediately skips to step 27:

```
20        feeder.1 = 1037
21        feeder.2 = 1038
22        .
23        WHILE SIG(feeder.1, feeder.2) DO
24            CALL move.part.1()
25            CALL move.part.2()
26        END
27
28    ; Either feeder #1 or feeder #2 is empty
29        .
30        .
31        .
```

Related Keywords

DO program instruction

EXIT program instruction

NEXT program instruction

UNTIL program instruction

WINDOW program instruction

Syntax

WINDOW %belt_var = location, location, program, priority

Function

Set the boundaries of the operating region of the specified belt variable for conveyor tracking.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The BELT switch must be enabled for this instruction to be executed.

The belt variable referenced must have already been defined using a DEFBELT instruction.

Parameters

%belt_var	Name of the belt variable whose window is being established.
location	Compound transformation that, together with the direction of the belt, defines one boundary of the operating window along the belt. The window boundaries are planes that are perpendicular to the direction of belt travel and include the positions specified by the two transformations. The order of the transformations is not important- this instruction automatically determines which transformation represents the upstream boundary and which is for the downstream boundary.
program	Optional program that is called if a window violation occurs while tracking the belt, subject to the specified priority level and the current priority level of the system.
priority	Optional priority level of the window violation program. If no priority is specified, a priority of 1 is set.

Details

The operating window defined by this instruction is used both at motion planning time and motion execution time to determine if the destination of the motion is within acceptable limits.

When a motion is being planned, the destination of the motion is compared against the operating window. If a window violation occurs, the window violation program is ignored and a program error may be generated depending upon the setting of the BELT.MODE parameter and the nature of the error.

When a motion relative to the belt is being executed or after the motion is completed and the robot continues to track the destination, the destination is compared against the window every eV+ trajectory cycle. If a window violation occurs and a program has been specified, the program is automatically invoked subject to its priority level, and the robot continues to track the belt and follow its continuous path motion. (The presumption is made that the specified program directs the robot as required to recover from the window violation.)

If no program has been specified, the robot is immediately stopped and a window violation program error is signaled. If a REACTE has been posted, the REACTE routine is activated. Otherwise, program execution is terminated.

Example

The working window for the belt variable %belt1 is defined by locations win1 and win2. If a window violation ever occurs while the robot is tracking the belt, the program belt.error is executed as a subroutine:

```
WINDOW %belt1 = win1, win2, belt.error
```

Related Keywords

BELT system switch

BELT real-valued function

BELT.MODE system parameter

BSTATUS real-valued function

DEFBELT program instruction

SETBELT program instruction

WINDOW real-valued function

WINDOW real-valued function

Syntax

WINDOW (transformation, time, mode**)**

Function

Return a value that indicates where the location described by the belt-relative transformation value is relative to the predefined boundaries of the working range on a moving conveyor belt.

Usage Considerations

This option is available only if your Adept system is equipped with the eV+ Extensions option.

The BELT system switch must be enabled before this function can be used.

The belt variable referenced in the compound transformation must have already been defined using a DEFBELT instruction.

Parameters

transformation	Compound transformation value that is defined relative to a belt. That is, the compound transformation must begin with a belt variable.
time	Optional real-valued expression that specifies the time to look ahead when the transformation is evaluated. That is, the result of the function is the value predicted to apply time seconds in the future, based on the current belt position and speed. This parameter is used to test whether a motion can be correctly completed within an anticipated time period. A time of zero (the default value) tests the instantaneous value of the location. Negative times are converted to 0 and times greater than 32,768/60 seconds are set equal to 32,768/60.
mode	Optional real-valued expression that specifies whether the result of the function represents a distance inside or outside the belt window (see below).

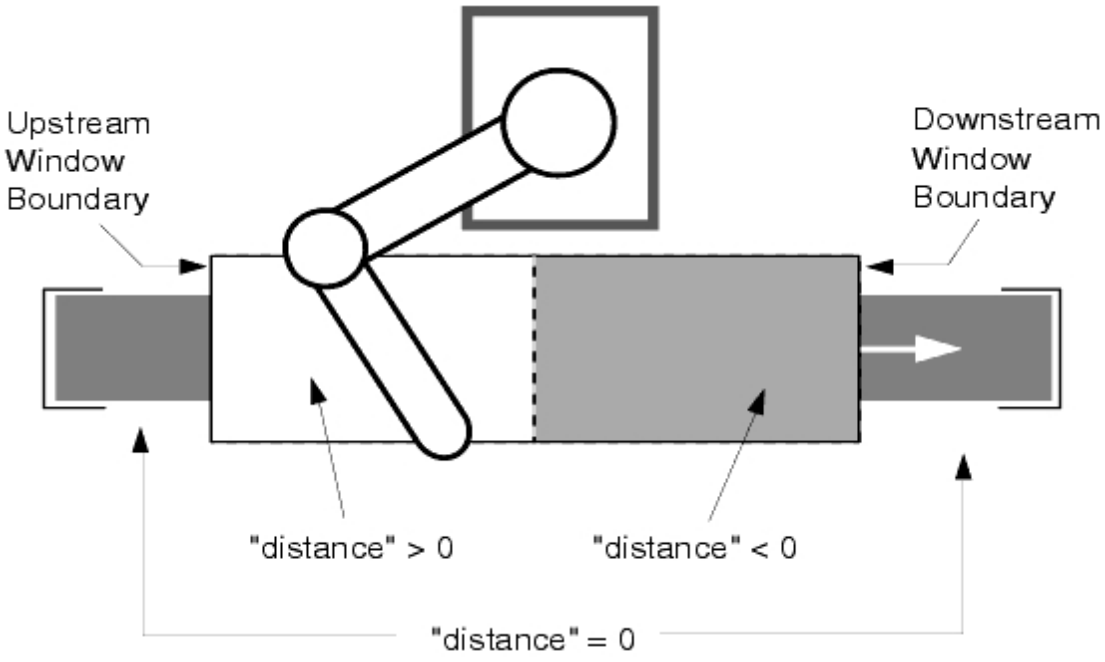
Details

The value returned, which is a distance in millimeters, should be interpreted as described below. (Note that the definitions of upstream and downstream depend on the value of the

BELT.MODE system parameter.)

1. If the value of the mode expression is *less than or equal to zero* (the default), the value returned is interpreted as follows (see the following figure):

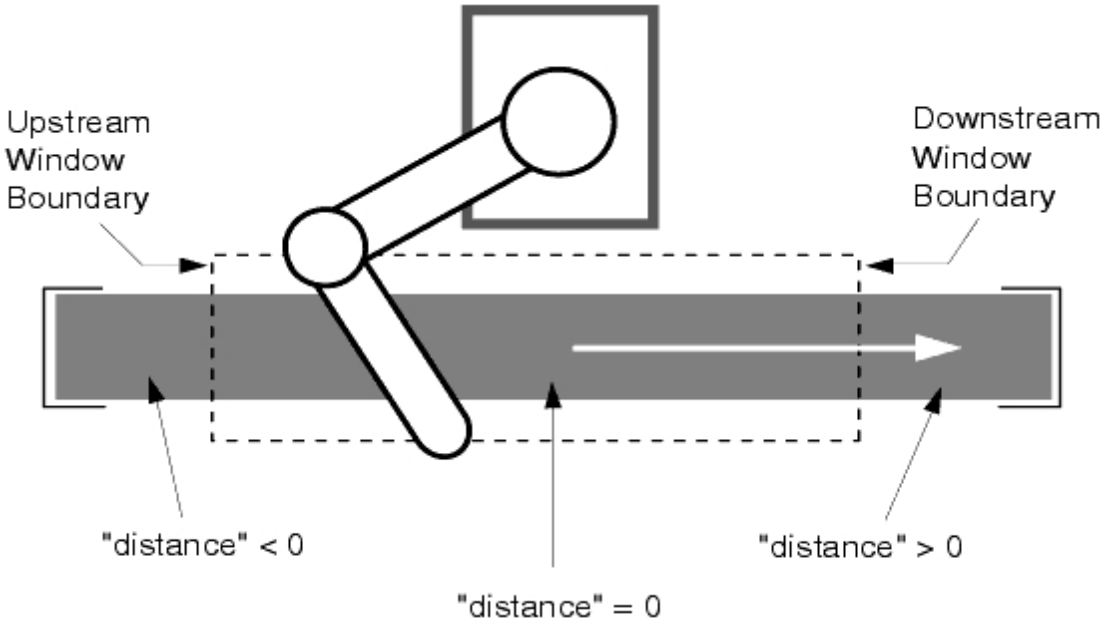
0	The location is outside the window.
<0	The location is inside the window, closest to the downstream window boundary; the distance is ABS(value_returned).
>0	The location is inside the window, closest to the upstream window boundary; the distance to the boundary is the returned value.



WINDOW Function for Mode < or = 0

2. If the value of the mode expression is *greater than zero*, the value returned is interpreted as follows (see the following figure):

0	Indicates the location is within the window.
<0	Indicates the location is upstream of the upstream window boundary; the distance is ABS(value_returned).
>0	Indicates the location is downstream of the downstream window boundary; the distance is the value returned.



WINDOW Function for Mode > 0

Note that the value returned by the WINDOW function always becomes more positive as the test location moves downstream (except for the discontinuity at the middle of the window when the mode value is less than or equal to zero).

Example

```
distance = WINDOW(%belt1:pick.up, 2, 1)
```

The distance is nonzero if, in two seconds, the location will be outside the operating window for %belt1. Otherwise, distance is zero if the location is within the window.

Related Keywords

BELT real-valued function

BELT system switch

BELT.MODE system parameter

BSTATUS real-valued function

DEFBELT program instruction

SETBELT program instruction

WINDOW program instruction

WRITE program instruction

Syntax

WRITE (lun,record_num**) format_list**

Function

Write a record to an open file, or to any I/O device. For a network device, write a string to an attached and open TCP connection.

Usage Considerations

Except for the monitor window or console serial port, the device to receive the output must have been attached. If the output is to a disk file, the file must have been opened with an FOPENA or FOPENW instruction.

Program execution waits for the write operation to complete unless there is a /N format specifier in the format list.

Parameters

lun	Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)
record_num	Optional real-valued expression that represents the number of the record to be written. This should be 0 (the default value) to write the next sequential record. If the value is not zero, the record is written in random-access mode (which requires that the records all have the same length). In random-access mode, records are numbered from one (to a maximum of 16,777,216). When accessing the TCP device with a server program, this parameter is an optional real value, variable, or expression (interpreted as an integer) that defines the client handle. For more information, refer to documentation for the READ instruction.
format_list	Consists of a list of output variables, string expressions, and format specifiers used to create the output record. The format list is processed exactly like an output specification for the TYPE instruction. When accessing the TCP device, you can include the /N specifier to prevent the eV+ system from waiting for a write acknowledgment.

Details

This is a general-purpose data output instruction that writes a record to a specified logical unit. A record can contain an arbitrary list of characters, but must not exceed 512 characters in length.

For files that are opened in fixed-length record mode, this instruction appends NULL characters to the output record if it is shorter than the file records.

When accessing the TCP/IP device, the record_num parameter enables a server to communicate with multiple clients on a single logical unit. Handles are allocated when a client connects to the server and deallocated when a client disconnects. During a connection the read instruction that receives data from the TCP logical unit returns the client handle. A write instruction can then use the handle value to send data to the corresponding client.

Examples

You can write a message to the manual control pendant with the following instructions:

```
ATTACH (1)                    ;Attach the control pendant
WRITE (1) $message            ;Output message to pendant
DETACH (1)                    ;Detach the pendant
```

A file with variable-length records can be written to the system disk drive with instructions such as the following:

```
ATTACH (dlun, 4) "DISK"          ;Attach the disk interface
FOPENW (dlun) "A:testfile.dat"   ;Open a file on drive "A"
FOR i = 0 TO LAST($lines[])      ;Loop for all the elements
     WRITE (dlun) $lines[i]      ;to be written
END
FCLOSE (dlun)                    ;Close the file
DETACH (dlun)                    ;Release the disk interface
```

Attach to serial line 1 and write a greeting:

```
ATTACH (slun, 4) "SERIAL:1"
WRITE (slun) "Hello from serial line 1"
```

Write the string $str to the client defined by the handle, which must have been defined previously when the a message was received. Do not wait for acknowledgment:

```
WRITE (lun, handle) $str, /N
```

Related Keywords

ATTACH program instruction

DETACH program instruction

FCLOSE program instruction

FEMPTY program instruction

FOPEN_ program instruction

IOSTAT real-valued function

PROMPT program instruction

READ program instruction

TYPE program instruction

XOR operator

Syntax

... XOR value ...

Function

Perform the *logical* exclusive-OR operation on two values.

Details

The XOR operator operates on two values, resulting in their logical exclusive-OR combination. For example, during the exclusive-OR operation

```
c = a XOR b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	->	FALSE
FALSE	TRUE	->	TRUE
TRUE	FALSE	->	TRUE
TRUE	TRUE	->	FALSE

That is, the result is TRUE if only *one* of the two operand values is logically TRUE. To review the order of evaluation for operators within expressions, see the section Order of Evaluation in the *eV+ Language User's Guide*.

Example

In the following sequence, the instructions immediately following the IF are executed when not_eq_one is TRUE and count does not equal 1, or when not_eq_one is FALSE and count equals 1. Otherwise, they are *not* executed.

```
IF not_eq_one XOR (count == 1) THEN
...
END
```

Related Keywords

AND operator

BXOR operator

OR operator

ID Option Words

The following topics are described in this chapter:

Introduction to ID Option Words **572**
Robot Option Words **572**
System Option Words **574**
Processor Option Word **576**

Introduction to ID Option Words

This appendix supplements the descriptions of the ID monitor command and the ID( ) real-valued function.

The ID command displays various option words as hexadecimal values; the ID function makes the same values available to programs. This appendix describes the following:

- Basic eV+ system (two option words)
- Processor option word for each processor

Robot Option Words

The real-valued functions ID(8,8) and ID(8,10+robot) return the first option word for the selected and specified robot, respectively. The interpretations of the high-order bits in this option word are described in the following table. Note that the interpretation of the bits in the high byte of this value (bits 9 through 16) is independent of the particular kinematic module being used. However, the bits in the low byte depend on the robot module.

Robot Option Word #1 (from ID(8, 10+robot)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
9	256	100	Motor limit-stop testing is enabled. This should be enabled only for robots that have excessive motor coupling, and that have motor limit-stop data allocated.
10	512	200	FREE mode power off is enabled. Any limit violations during FREE mode cause robot power to be disabled.
11	1024	400	The system will automatically execute a CALIBRATE monitor command when the system is booted.
12	2048	800	Obsolete
13	4096	1000	Check for collisions with static obstacles in Cartesian space during joint-interpolated motions. (This bit applies only to robot modules that can perform straight-line motions. Thus, for example, it does not apply to the JTS device

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
			module.) Note that when this bit is set, joint-interpolated motions have the same computational load as straight-line motions.
14	8192	2000	Micron display mode is enabled.
15-16			Reserved for future use (currently zero).

The real-valued functions ID(11,8) and ID(11,10+robot) return the second option word for the selected and specified robot, respectively. The interpretations of the bits in this option word are described in the following table.

Robot Option Word #2 (from ID(11, 10+robot)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
1	1	1	Robot has an RSC.
2	2	2	Robot has an extended-length quill.
3	4	4	Robot has the cleanroom option.
4	8	8	Whenusing the Adept Cobra s600, s800 or s800 Inverted robots: Robot has the joint 4 quill encoder disabled.
5	16	10	Robot has the high-torque option.
6	32	20	Model-specific option, interpreted as follows: **Robot Model** / **Meaning of Bit Set** Adept Cobra s800 Inverted / Robot has the IP65 option

<table>
<tr><th rowspan="2">Bit #</th><th colspan="2">Mask Value</th><th rowspan="2">Interpretation When Bit Set</th></tr>
<tr><th>Decimal</th><th>Hexadecimal</th></tr>
<tr><td>7</td><td>64</td><td>40</td><td>Robot has the EC certification option.</td></tr>
<tr><td>8</td><td>128</td><td>80</td><td>Robot has high-resolution joint 4.</td></tr>
<tr><td>9</td><td>256</td><td>100</td><td>Model-specific option, interpreted as follows:<table><tr><th>Robot Model</th><th>Meaning of Bit Set</th></tr><tr><td>Adept Cobra s600/s800</td><td>Robot is inverted</td></tr></table></td></tr>
<tr><td>10</td><td>512</td><td>200</td><td>Model-specific option, interpreted as follows:<table><tr><th>Robot Model</th><th>Meaning of Bit Set</th></tr><tr><td>Adept Cobra s600/s800</td><td>Robot has amber LED (for UL conformance)</td></tr></table></td></tr>
<tr><td>11-16</td><td></td><td></td><td>Reserved for future use (currently zero).</td></tr>
</table>

System Option Words

The configuration of a specific eV+ system can be determined by examining two "option words" that are displayed (as hexadecimal numbers) when the system is loaded from the system boot disk, and by the ID monitor command. The values of the option words are also available to programs from the real-valued functions ID(5) and ID(6).

The option words should be interpreted as bit fields, which indicate information about the system configuration. The interpretations of the bits in the first system option word [returned by the function ID(5)] are described in the following table.

System Option Word #1 [from ID(5)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
1	1	1	"eV+ Extensions" software license installed
2	2	2	External encoders are supported. (See Note 1.)
3	4	4	Robot or motion devices are enabled
4-7			Reserved for future use (currently zero)
8	128	80	ALTER instruction enabled. (See Note 2.)
12	2048	800	MOVEF and MOVESF instructions enabled

NOTES:
1. The External encoder bit is used with robot systems to indicate the "conveyor tracking" capability. With non-robot systems it is used to indicate the "external encoder option".

2. This bit tracks the "eV+ Extensions" bit.

The interpretations of the bits in the second software option word [returned by the function ID(6)] are described in the following table.

System Option Word #2 [from ID(6)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
1-5			Reserved for future use (currently zero)
6	32	20	Vision is enabled
7	64	40	Guidance vision is enabled
8	128	80	Inspection vision is enabled

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
9	256	100	DDCMP option is installed. (See Note 1.)
10-12			Reserved for future use (currently zero)
13	4096	1000	Network hardware is installed, and TCP is installed. You can use the NETWORK real-valued function to obtain detailed information about the network.
14	8192	2000	EC Safety Category 1 system
15	16384	4000	EC Safety Category 3 system
16			Reserved for future use (currently zero)
NOTES: 1. This bit tracks the "eV+ Extensions" bit in the first option word.			

Processor Option Word

The interpretations of the bits in the processor option word [returned by the function ID(6, 4)] are described in the following table.

Processor Option Word [from ID(6, 4)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	Hexadecimal	
1	1	1	Processor is running the eV+ Operating System
2	2	2	Processor is running the Vision processing software
3	4	4	Processor is running the Servo software
4-16			Reserved for future use (currently zero)

System Messages

The following topics are described in this chapter:

Introduction to System Messages **578**
System Messages - Alphabetical List **578**
SystemMessages - Numerical List **653**

Introduction to System Messages

While the eV+ system is being used, it is possible for hardware and software errors to occur. For example, if commands or instructions are not entered in the correct way, eV+ rejects the input. The usual response is to write an error message to the system terminal indicating what is wrong so that you can correct the error. In addition to error messages, eV+ can also issue warning or informational messages.

In eV+, each error message is normally assigned an error number (or code) and an associated error string. The following numbering conventions are used to identify the type of system message generated:

- Informational Messages (numbers 0 to 49) list messages that provide information.
- Warning Messages (numbers 50 to 299) list warning messages that you may receive.
- Error Messages (negative numbers) list the error messages that you may receive.

Most message codes associated with errors can be made available to a program by the ERROR function, which returns the code of the latest error that occurred. In addition, the $ERROR function returns the error message associated with any eV+ error code.

Documentation

The eV+ system message documentation allows you to look up the details of an error message either alphabetically by the message string, or numerically by the system code. For details, see Related Topics.

For each message, the documentation includes the message code, the text of the message, and sometimes a comment about the applicability of the message. Angle brackets (<...>) are used to enclose a description of an item that appears in that position. All numbers are decimal.

Updates

Changes to eV+ system messages are summarized in the eV+ Release Notes.

Related Topics

Alphabetical List of eV+ System Messages

Numerical List of eV+ System Messages

ERROR real-valued function

$ERROR string function

System Messages - Alphabetical List

Almost every eV+ system message has a numeric code that can be used to identify the message within a eV+ program. The ERROR and IOSTAT functions return this code. The $ERROR string function returns the error message corresponding to an error code.

All of the eV+ messages are described in this section. Each description includes the text of the message, its error code if applicable, an explanation of the likely cause of the message, and a suggestion of what action you should take. The error code for each message is listed after the error text, for all those messages that have a code.

The system messages are arranged alphabetically by system message text. Click an underlined letter to jump to the first message that begins with that letter. For a list of the system messages sorted by numerical codes, see System Messages - Numerical List.

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

> **NOTE:**If the system has more than one robot connected and an error is associated with a specific robot, the robot number is appended to the error message in the form (Robot #).

1394 communications timeout (-927)

Explanation:eV+ has timed out waiting for a node on the 1394-based servo network to respond. A transient network problem may have interfered with normal communications, or the network node itself may have failed.

User action: Retry the operation that failed. Reinitialize your servo network. Reboot your robot controller. If the problem persists, contact Adept Customer Service.

Adept Digital Workcell Simulation mode (None)

Explanation: This message is displayed during eV+ initialization (i.e., during the booting process), and as part of the output displayed by the ID monitor command, to indicate that the eV+ system is operating in Adept Digital Workcell simulation mode.

User action: No response is required if the eV+ system is intended to be used in Adept Digital Workcell simulation mode. Otherwise, the command DISABLE ADW should be entered and the Adept controller should be rebooted.

Aborted (-400)

Explanation: The last command requested, or the program that was executing, has been aborted at your request.

User action: None.

Already attached to logical unit (-515)

Explanation: A program has executed more than one ATTACH instruction for a specific logical unit, without executing a DETACH in between. (The program is still attached to the logical unit after this error occurs.)

User action: Check the program logic and remove redundant ATTACH instructions.

Ambiguous AUTO invalid (-477)

Explanation: When exiting from the program editor, eV+ has encountered an automatic variable with undetermined type. That is, the system cannot determine if the variable is real-valued or a transformation. Automatic variables cannot be ambiguous, since their storage requirements must be known before they are referenced.

User action: Include the REAL or LOC type specification parameter in the AUTO statement that declares the variable, or reference the variable in a program instruction in a manner that makes its type clear.

Ambiguous name (-453)

Explanation: The abbreviation used for the last command, instruction, or system-defined name was not long enough to identify the operation intended.

User action: Reenter the last line, using a longer abbreviation.

Are you sure (Y/N)? (10)

Explanation: The requested command has a significant effect on the state of the system, and eV+ wants to know if you really want it to happen.

User action: To have eV+ continue, type **y** followed by a carriage return. An **n** followed by a carriage return or just a carriage return causes the command to be aborted.

Arithmetic overflow (-409)

Explanation: The result of a calculation was outside the allowable range for real variables or eV+ has encountered a number that is outside the allowed range for integers while converting a real-valued number to a decimal, hexadecimal, or octal integer, or logical value. Logical values use 32-bit integers, but most program instructions that require integer arguments allow only 16-bit integers. Also, real variables can have only magnitudes in the range from about 5.4E-20 to 9.2E+18.

User action: Modify the program as required.

Attempt to modify active belt (-614)

Explanation: A program instruction has been executed that modifies the belt variable that is currently being tracked by the robot.

User action: Change the program in order not to modify the variable while the robot is tracking it.

Attempt to redefine variable class variable_name (-470)

Explanation: Upon exiting from the editor, the named variable was found in two of the following places: the .PROGRAM argument list, an AUTO statement, a LOCAL statement, or a GLOBAL statement.

User action: Modify the program to include the variable in only one of these places.

Attempt to redefine variable type variable_name (-469)

Explanation: If a program is being edited, the line just entered contains a reference to a variable in a manner inconsistent with its use elsewhere in the program. The most likely problem is confusing a location variable with a real variable. If you just exited from the editor, the named variable conflicts with a global variable that already exists.

User action: If the new use of the variable is correct, you must delete all references to the incorrect variable and then reenter the statement that caused the error. If the new use is incorrect, use a different variable name. If there is a conflict with a global variable, either use a DELETE_ command to delete that variable, or make the conflicting variable AUTO or LOCAL to the current program.

Auto Startup... (None)

Explanation: The automatic start-up procedure has begun. (See the discussion of command programs for more information.)

User action: None required for this message, but subsequent commands in the auto-startup command program may require user action.

WARNING: The robot may begin to move during the automatic start-up procedure. If necessary, you can stop the robot by pressing EMERGENCY STOP or the controller or PANIC on the pendant.

Backplane E-STOP detected by CPU (-630)

Explanation: The AdeptMotion system has detected an error or problem and has asserted the BRKSTOP signal on the VME bus. If that error is seen, it indicates a transient BRAKE-ESTOP signal.

User action: Correct the problem that is causing the motion system to report the error.

Bad block in disk header (-523)

Explanation: While formatting a disk, a bad disk block has been found in the disk header area. The format operation has failed, and the disk is not usable.

User action: Enter the FORMAT command again-use a different diskette if the error persists.

Belt not enabled (-615)

Explanation: A robot operation that references a moving conveyor belt has been attempted when the conveyor tracking feature is disabled.

User action: Enter an ENABLE BELT command and retry the operation.

Belt servo dead (-617)

Explanation: The belt processor isn't responding to commands from eV+.

User action: After saving the programs, power down the controller and power it up again. If this error occurs repeatedly, contact Adept Customer Service.

Belt window violation (-616)

Explanation: Either a robot motion has been planned that moves the robot outside of the belt window, or the robot has moved outside of the belt window while tracking the belt.

User action: Modify the program so that the robot does not move outside the belt window. Consult the BELT.MODE parameter and the WINDOW instruction for different ways to define the belt window.

Branch to undefined label Step nnn (-412)

Explanation: A program instruction references a program label that is not defined in the program. Either the label is missing or was mistyped when defined or in the reference.

User action: Check the label definition and reference.

Breakpoint at (task) program_name, step n (17)

Explanation: A breakpoint was encountered before the indicated step. (Any output associated with the breakpoint is displayed after the message shown above.)

User action: Enter a PROCEED (Ctrl+P), RETRY, SSTEP (Ctrl+Z), or XSTEP (Ctrl+X) command to resume program execution.[1] Otherwise, enter any other monitor command.

Breakpoint not allowed here (-380)

Explanation: An attempt has been made to set a breakpoint before the first executable statement of a program.

User action: Enter a new BPT command specifying a step after the first executable statement. That is, after the .PROGRAM statement, any AUTO and LOCAL statements, and all comments and blank lines at the start of the program.

Calibration program not loaded(-425)

Explanation: A program required for calibration has not been loaded from disk. This error occurs if a robot-specific calibration file cannot be found on the disk (i.e., in the same location as the file for the main calibration program), or if a required calibration program is not present in memory when it is expected. The latter situation can occur if the CALIBRATION command or instruction is executed with an input mode that does not cause the calibration programs to be loaded from disk, and the programs are not already present in memory.

User action: Reissue the CALIBRATE command or instruction with the proper mode. The default mode of zero causes CALIBRATE to automatically load the required programs from disk, perform the calibration, and then delete the programs. Alternatively, issue a CALIBRATE command or instruction with mode "1" (which causes the calibration programs to be loaded into memory), and then reissue the CALIBRATE command or instruction that was originally attempted.

Calibration sensor failure Mtr n (-1106)

Explanation: During calibration, the calibration sensor for the indicated motor cannot be read correctly. Either the robot is blocked from moving, or a hardware error has occurred.

User action: Retry the CALIBRATE command or instruction after making sure that the robot is not blocked. If the problem persists, contact Adept Customer Service.

Canceled (-358)

Explanation: An editor, debugger, or pendant operation has been terminated due to operator intervention.

User action: This is usually an informative message to acknowledge the cancellation of the operation.

Can't access protected or read-only program (-310)

Explanation: An attempt has been made to edit a protected or read-only program. These programs cannot be edited.

User action: None.

Can't ALTER and track belt (-626)

Explanation: Either a belt-relative motion was specified while ALTER mode was enabled, or an attempt was made to enable ALTER mode while the selected robot was tracking a belt.

Both operations are prohibited because belt-tracking and ALTER mode cannot be performed at the same time.

User action: Either disable ALTER mode or stop tracking the belt.

Can't change modes while task running (-361)

Explanation: A command was issued to change from debug monitor mode to debug editor mode while the program task being debugged was executing. You can change to debug editor mode only when the associated task is stopped.

User action: Stop execution of the program task being debugged, or continue without using debug editor mode.

Can't create new slide bar (-557)

Explanation: An attempt has been made to create a graphic slide bar in the horizontal or vertical scroll bar. Slide bars should be created only in the main window, although they can appear in the title or menu bars.

User action: Modify the arguments for the GSLIDE instruction to have the slide bar created within the window.

Can't create program in read-only mode (-364)

Explanation: An attempt has been made to initiate editing of a program in read-only access mode, but the program does not exist.

User action: If the program name was entered incorrectly, enter the command again with the correct name. Do not select read-only access (with /R) when creating a new program.

Can't delete .PROGRAM statement (-350)

Explanation: An attempt has been made to delete the .PROGRAM statement while editing a program.

User action: To change the .PROGRAM statement, replace it with another .PROGRAM statement. To delete lines at the beginning of the program, move down to line 2 before issuing delete commands.

Can't execute from SEE program instruction (-362)

Explanation: An attempt has been made to use a SEE editor command that cannot be used after the editor has been initiated with the SEE program instruction.

User action: Enter another command or exit the editor and reenter from the eV+ monitor.

Can't exit while lines attached (-355)

Explanation: You attempted to terminate execution of the editor while lines were present in the attach buffer. The attach buffer must be empty before the editor can be exited.

User action: You can use Shift+Copy to deposit the contents of the attach buffer into the current program. You can also use Esc+K to delete lines from the attach buffer (99 Esc+K deletes up to 99 lines from the buffer).

Can't find calibration program file (-426)

Explanation: While processing a CALIBRATE command or instruction, the eV+ system cannot find the calibration utility program file CAL_UTIL.V2.

User action: Restore the missing file from the eV+ distribution disk to the current default directory, or to the directory \CALIB\ on the local "C" or "D" drive.

Can't go on, use EXECUTE or PRIME (-313)

Explanation: An attempt has been made to continue the execution of a program that has completed or stopped because of a HALT instruction. Normally, an error results when a PROCEED, RETRY, or XSTEP command is entered (or the pendant RUN/HOLD button is pressed) after a program has completed all its cycles.

User action: Use the EXECUTE or PRIME command, or the pendant PRIME function, to restart the program from the desired instruction.

Can't interpret line (-450)

Explanation:eV+ cannot interpret the last command or instruction entered.

User action: Verify the spelling and usage, and reenter the line. In the case of an error while loading from the disk, edit the affected programs to correct the indicated lines-they have been converted to bad lines.

Can't mix MC & program instructions (-414)

Explanation: A program instruction has been encountered during processing of a command program, or an MC instruction has been encountered in a normal program.

User action: Edit the command program to use the DO command to include the program instruction, or remove the MC instruction from the normal program.

Can't start while program running (-312)

Explanation: An attempt has been made to start execution of a program from the pendant while a program is already executing as task #0.

User action: Stop the program currently executing and then retry the operation.

Cartesian control of robot not possible (-635)

Explanation: A program has attempted to perform a straight-line motion with a robot that does not support such motions.

User action: Change the program to use joint-interpolated motion.

Change? (11) Change

Explanation: You are being given an opportunity to modify the location value just created by a HERE or POINT command.

User action: Enter any desired new components, separated by commas, or press the Return key to indicate that no changes are desired.

..., change to: (None) Change

Explanation: While initiating a string replacement operation, the SEE editor is prompting for the string to be used for the replacement.

User action: Enter the desired replacement string. Note that if you press Return, the string to be searched for is erased (that is, an empty string is used for the replacement).

Collision avoidance dead-lock (-647)

Explanation: Two robots with collision detection enabled are simultaneously blocking each other's path. That is, neither robot can perform its next motion until the other robot moves out of the way.

User action: Change the application program to prevent the deadlock situation.

Command? (None)

Explanation: A SEE editor extended command has been initiated with the X command.

User action: Enter the desired extended command, or press Return to cancel the request.

Communication time-out (-531)

Explanation: An I/O operation has not completed within the allotted time interval. For data communications, the remote communications device has not properly acknowledged data that was sent.

User action: Make sure the remote device is communicating. Make sure connections to the remote device are operating properly.

Communications overrun (-524)

Explanation: Data has been received on an I/O device faster than eV+ is processing it, and some data has been lost. This happens only on the serial interface line or the network.

User action: Modify the program to service the I/O device more often, add a handshaking protocol, or slow down the transmission rate to eV+.

COMP mode disabled (-603)

Explanation: The command attempted requires computer control of the robot, but COMPUTER mode was not selected on the pendant.

User action: Select COMP mode on the pendant or enable DRY.RUN mode from the terminal, then reissue the command.

Connecting to Adept Digital Workcell system via Ethernet (^C to quit) (None)

Explanation: This message is displayed during eV+ initialization (i.e., during the booting process) to indicate that the eV+ system is operating in Adept Digital Workcell simulation mode, and that the eV+ system is waiting for a response from the PC that is running the Adept Digital Workcell software.

User action: No response is needed if the connection completes successfully. If the connection does not complete, you can enter CTRL+C to cancel waiting for the connection. The eV+ system then reports failures of all the robot servos and all the installed licenses are disabled (except those needed to use the Adept ACE user interface). In that case, you should either try again to start up the Adept controller (after making sure that the Adept Digital Workcell software is running on the PC), or enter the eV+ monitor command DISABLE ADW and restart the Adept controller.

Controller not in automatic mode (-303)

Explanation: An attempt has been made to initiate program execution or PRIME a program from the monitor window or command terminal when the controller is not in automatic mode.

User action: Select automatic mode by moving the switch on the front panel to the automatic position, or by activating the proper switch on a custom control panel. Retry the previous command.

Controller not in manual mode (-304)

Explanation: An attempt has been made to perform an operation that requires the controller to be in manual mode when it is not in manual mode. If you do not have a front panel connected, the controller is assumed to be in automatic mode.

User action: Select manual mode by moving the switch on the front panel to the manual position, or by activating the proper switch on a custom control panel. Retry the previous operation.

Controller not in network mode (-317)

Explanation: An attempt has been made to use a serial line configured for network use, but the controller is not in network mode. If you do not have a front panel connected, the controller is assumed to be in local mode.

User action: Select network mode by moving the switch on the front panel to the network position, or by activating the proper switch on a custom control panel. Retry the previous operation.

Control structure error (-473)

Explanation: An incomplete control structure has been encountered during program execution.

User action: Edit the program to correct the control structure.

*Control structure error * Step nn (-472)

Explanation:eV+ has detected an incomplete or inconsistent control structure at the specified step when exiting the program editor, loading a program, or processing a BPT command.

User action: Edit the program to correct the control structure. (Note that the actual error may not be at the indicated step.) If the error occurs in response to a BPT command, you can type **dir /?** to identify programs that are not executable and thus might contain the control-structure error.

Cursor at column n (None)

Explanation: The SEE editor WHERE extended command is reporting the current column position of the cursor.

User action: None. This is an informational message.

Database manager internal error (-859)

Explanation: This error indicates that the system has encountered an inconsistency.

User action: Contact Adept Application Engineering. Please record the details of exactly what you were doing at the time the error occurred.

Data checksum error (-510)

Explanation: An error was detected while transferring information to or from an external device.

User action: Attempt the transfer again. If the problem persists, contact Adept Customer Service.

Data error on device (-522)

Explanation: An error was detected while attempting to read information from an external device, possibly because a diskette has been damaged or was not formatted properly.

User action: Attempt the read again. Make sure the correct diskette is being used, that it is properly installed in the drive, and that it is formatted. (Recall that formatting a diskette erases its contents.)

Device error (-660)

Explanation: An error was detected for an external device such as one specified in the last DEVICE or SETDEVICE program instruction. The actual error depends upon the type of device referenced.

User action: Make sure the instruction's parameters are valid. Refer to the documentation for the device type referenced for information on how to determine what has caused the error.

Device full (-503)

Explanation: There is no more space available on a device. If this error is received for a disk file, it indicates that the disk is full (if there are many small files on the device, this error indicates the directory is full). If this error is received for a servo device, it indicates that an attempt has been made to assign too many servo axes to a single CPU.

User action: Delete unneeded disk files, or use another drive or diskette. Reconfigure your system so the maximum number of axes per CPU is not exceeded.

Device hardware not present (-658)

Explanation: An attempt has been made to reference a device that is not present in your system.

User action: Verify that the device was correctly specified. Verify that the device hardware is present and is configured properly.

Device in use (-668)

Explanation: An attempt has been made to attach, assign, or configure a hardware device (e.g., a VMI axis) that is already being used.

User action: Check the program code to make sure the requested device has not already been attached. If the error occurs during power-up initialization of the eV+ system, enter the SRV.NET monitor command and look for multiple nodes that are reporting the same input or output block number.

Device not ready (-508)

Explanation: (1) The requested disk device (or remote network task) is not prepared to communicate with the eV+ system.

(2) A limited-access device like the terminal, the pendant, or a serial line is attached to a different program task.

(3) You have tried to write into a pull-down window while it is displayed.

User action: (1) If the intended device is a system microfloppy disk drive, make sure the diskette is correctly inserted and formatted.

(2) If a limited-access device is specified, ABORT and KILL the program task that has it attached, or wait for the program task to release the device. If the intended device is on the network, verify that the proper connections are made and that the remote system is operating correctly. (2) ABORT and KILL the program task that has the device attached, or wait for the task to release the device.

Device reset (-663)

Explanation: The device is busy processing a reset operation. The reset can have been requested (with a SETDEVICE instruction) by another program task that is accessing the device, or the device can have initiated the reset on its own.

User action: Use software interlocks to prevent a second program task from accessing the device after a reset operation has been requested. (Note that the requesting SETDEVICE instruction waits for the reset to complete.) Refer to the documentation for the specific device for information on its self-generated resets.

Device sensor error (-662)

Explanation: A hardware error occurred in the sensing system accessed with the last DEVICE instruction.

User action: Refer to the documentation for the sensing system for information on how to determine the cause of the error.

Device time-out -659)

Explanation: The device has not responded within the expected time.

User action: Check the documentation for the device type referenced to determine what caused the error. Verify that the device hardware is configured properly.

Directory error (-509)

Explanation: This error can occur when performing a READ instruction (following an FOPEND instruction).

User action: Unlike most other errors, this error can be ignored. Additional READ instructions to the same directory correctly return additional contents of that directory.

Directory not empty (-571)

Explanation: The operation attempted to remove a directory that was not empty.

User action: Delete the directory's contents before deleting the directory.

DO not primed (-302)

Explanation: A DO command was attempted without specifying a program instruction to be executed and no previous DO had been entered.

User action: Provide the desired instruction with the DO command.

Driver internal consistency error (-519)

Explanation: An I/O device or servo has responded in an unexpected manner.

User action: Retry the operation that caused the error. If it persists, contact Adept Customer Service.

Duplicate .PROGRAM arguments (-468)

Explanation: At least two of the arguments in a .PROGRAM statement have the same name.

User action: Edit the .PROGRAM line so that all the arguments have unique names.

Duplicate servo node ID (-678)

Explanation: During startup, or after a servo bus reset, eV+ has detected two or more servo nodes with the same serial number. Either an error has occurred with servo node detection, or a servo node is configured incorrectly. Servo network operation is not allowed.

User action: Use the SRV.NET command to determine which nodes are causing this error. Reinitialize your servo network. Reboot your robot controller. If a new servo node has been added to the network, verify that its serial number is correct. If the problem persists, contact Adept Customer Service.

Duplicate statement label (-464)

Explanation: The same program statement label is used more than once in a user program.

User action: Change one of the duplicate labels.

Duty-cycle exceeded Mtr n (-1021)

Explanation: The indicated motor has been driven fast for too long a period of time. The servo system has disabled Arm Power to protect the robot hardware.

User action: Turn on Arm Power; reduce the speed and/or acceleration for the motion that was in progress or for motions that preceded that motion; and repeat the motion that failed.

Encoder fault Mtr n (-1025)

Explanation: The servo board has detected a broken encoder wire on the indicated axis.

User action: Inspect the encoder wiring for intermittent connections or broken wires. Try swapping the encoder cable with another. You can disable this error with the SPEC utility, but do so only as a last resort.

Encoder quadrature error Belt n (-1013)

Explanation: The position encoder signal from the specified conveyor belt is sending information that is not phased correctly. The encoder or its cabling may be defective. (Encoder error checking is initiated by the DEFBELT instruction and by enabling the BELT switch while a belt is defined.)

User action: Make sure the encoder cable is properly connected. Try to run the conveyor at a slower speed. Contact Adept Customer Service if the error persists.

Encoder quadrature error Mtr n (-1008)

Explanation: The position encoder signal from the specified motor is sending information that is not phased correctly. The encoder or its cabling may be defective.

User action: Turn on high power, calibrate the robot, and try to perform the motion at a slower speed. If the error persists, contact Adept Customer Service.

E-STOP 1 detected by CPU (-608)

Explanation: An E-STOP condition on E-STOP channel 1 has been detected by the CPU. Normally, this message is suppressed and the cause of the E-STOP is reported instead.

User action: If safe to do so, attempt to enable HIGH POWER and note any different error messages which occur. Verify that the CIP is connected securely. If this error occurs frequently, contact Adept Customer Service.

E-STOP 2 detected by CPU (-914)

Explanation: An E-STOP condition on E-STOP channel 2 has been detected by the CPU. Normally, this message is suppressed and the cause of the E-STOP is reported instead. There may be a hardware problem with the CIP, its cabling, or the AWC.

This error can result from redundant E-STOP channels that do not track each other within a reasonable time.

User action: Hold the MCP enable switch and reenable HIGH POWER as desired. If this error persists, contact Adept Customer Service.

E-STOP asserted by CPU (-919)

Explanation: An E-STOP condition has been generated by eV+ in response to an internal error condition. This error should not normally be seen.

User action: If safe to do so, attempt to enable HIGH POWER and note any different error messages that occur. If this error persists, contact Adept Customer Service.

E-STOP channels 1 and 2 do not match (-922)

Explanation: An E-STOP condition has occurred because the two redundant E-STOP signal channels do not report the same E-STOP state.

This error can be caused if redundant E-STOP channels do not track each other within a reasonable time (around 0.5 sec).

User action: If an external E-STOP circuit is being used, verify that both channels are wired and functioning properly.

E-STOP circuit is shorted (-923)

Explanation: An E-STOP condition has occurred because a short circuit in the E-STOP wiring has been detected.

User action: Verify that all E-STOP channels are wired and functioning properly and that there is no external power source wired to the E-STOP circuit. Check that the E-STOP source jumper on any connected robot is on the EXT position. If the problem persists, contact Adept Customer Service.

E-STOP circuit relay failure (-907)

Explanation: An E-STOP condition has occurred because the E-STOP chain on one channel is not in the same state as the other channel. This can be the result of welded contacts on

one of the two E-STOP Relays , a short circuit in one of the E-STOP channels , or the result of a pair of contacts connected to the E-STOP channels being in the opposite state (one open, the other closed).

User action: Check all user connections to each of the E-STOP channels (plugs fully seated, contacts on each channel in the same closed state). If no problem is uncovered in the user connections, call Adept Customer Service.

E-STOP detected by robot (-643)

Explanation: The motion interface board has detected an E-STOP due to the BRAKE-ESTOP signal being asserted on the VMEbus.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number of the task that received the error. If no additional information is available, call Adept Customer Service.

E-STOP from amplifier (-641)

Explanation: The SmartController has detected an E-STOP condition generated by the motor amplifiers. It indicates that the amplifiers have detected some fault condition.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number of the task that received the error.

E-STOP from front panel button (-908)

Explanation: An E-STOP condition has occurred because the E-STOP button on the front panel has been pressed.

User action: Unlatch the locking E-STOP button. Re-enable HIGH POWER as desired.

E-STOP from front panel external input (-911)

Explanation: An E-STOP condition has occurred because it was requested through the front panel external input signal.

User action: Restore the front panel external input signal state. Re-enable HIGH POWER as desired.

E-Stop from Line E-Stop input (-929)

Explanation: An E-STOP condition has occurred because it was requested through the Line E-STOP input signal.

User action: Restore the state of the Line E-STOP input signal. Re-enable HIGH POWER as desired.

E-STOP from MCP enable switch (-913)

Explanation: An E-STOP condition has occurred because the enable switch (formerly called Hold-to-Run switch) on the Pendant has been released. During MANUAL mode, releasing this switch performs a controlled power-off rather than an E-STOP.

User action: Hold the Pendant enable switch and re-enable HIGH POWER as desired.

E-STOP from MCP E-STOP button (-909)

Explanation: An E-STOP condition has occurred because the E-STOP button on the Pendant has been pressed.

User action: Unlatch the locking E-STOP button. Re-enable HIGH POWER as desired.

* E-STOP from safety system* Code n (-1111)

Because these message codes are related to hardware, refer to your Robot Instruction Handbook as your primary source of information. If it does not answer your questions, contact Adept Customer Service. The following table summarizes information about the codes.

MMSP External E-STOP Error Message Codes

Code n	Explanation
0	Adept E-stop, channel 1 error
1	Adept E-stop, channel 2 error
2	Customer E-stop, channel 1 error
3	Customer E-stop, channel 2 error

E-STOP from robot (-640)

Explanation: The motion interface board has detected an E-STOP condition generated by the RSC in the robot. This error is probably due to low air pressure, joint-1 overtravel, or motor overheating. A subsequent error message may provide more information.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number of the task that received the error. If no additional information is available, check for low air pressure, joint 1 overtravel, or motor overheating.

E-STOP from user enable switch (-912)

Explanation: An E-STOP condition has occurred because the user enable switch (formerly called the Hold-to-Run switch) has been released. During MANUAL mode, releasing this switch performs a controlled power-off rather than an E-STOP.

User action: Hold the user enable switch and re-enable HIGH POWER as desired. If the switch was already pressed, check the switch and the associated connectors and wiring.

E-STOP from user E-STOP button (910)

Explanation: An E-STOP condition has occurred because the user E-STOP button circuit has been broken.

User action: Restore the user E-STOP button circuit. Re-enable HIGH POWER as desired.

E-STOP from user muted safety gate (-921)

Explanation: An E-STOP condition has occurred because the user muted safety gate has been opened during automatic mode. During MANUAL mode, this error should not be seen.

User action: Close the muted safety gate and re-enable HIGH POWER as desired. If the gate was already closed, check the switch, the associated connectors, and wiring.

Executing in DRY.RUN mode (50)

Explanation: The DRY.RUN switch is enabled and program execution has been requested. Thus, no motion of the robot occurs.

User action: None unless motion of the robot is desired. In that case, abort execution of the program and disable the DRY.RUN switch.

Expected servo node not found (-679)

Explanation: After a servo bus reset, eV+ has not detected all the required nodes on the servo network. Either the network has failed because of noise, a servo node has failed, or a servo node has been unplugged.

User action: Use the SRV.NET command to determine which node is causing this error. Verify that all servo network connectors are secure. Reinitialize your servo network. Reboot your robot controller. If the problem persists, contact Adept Customer Service.

[Fatal] Addr Err at aaaaaa m:n I=xxxx, A=aaaa, F=ff (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Bus Err at aaaaaa m:n I=xxxx, t=aaaa, F=ff (None)

Explanation: A computer error occurred because of a bad read from memory, because of noise on the internal data bus, or because of a hardware problem.

User action: To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system. If the problem persists, contact Adept Customer Service.

[Fatal] CHK Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] DIV Instr Err at aaaaaa m:n (None)

Explanation: The eV+ system has detected an error from a divide instruction. This indicates a processor fault.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

[Fatal] Emul 1010 Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Emul 1111 Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] E-STOP signals are stuck off (-904)

Explanation: During system startup, a test is performed to ensure that no E-STOP signals are stuck in the off state. This error message is followed by one or more standard E-STOP error messages that indicate which signals are stuck. If this error occurs, robot power cannot be enabled.

User action: Check the wiring of your E-STOP circuits. Verify that the Controller Interface Panel (CIP) is connected properly. Contact Adept Customer Service for assistance.

[Fatal Force Err] ... ... (None)

Explanation: The force processor has detected an error condition. You can continue to use the eV+ system, but the force-sensing system cannot be used until you act upon the error.

User action: None required if you do not intend to use the force-sensing system. Otherwise, refer to the documentation for the force-sensing system for information on how to respond to the error.

[Fatal] Graphics/display processor error (None)

Explanation: The graphics processing unit on the graphics system processor has failed to respond to commands from the eV+ system.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

[Fatal] Illeg Instr at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Initialization failure Mtr n (-1014)

Explanation: During initialization of a robot kinematic module, the indicated motor failed initialization. The problem may be a missing or improperly configured servo interface board, or an incorrect motor mapping for this module.

User action: Verify that all servo interface boards are correctly installed and configured (use the SPEC.V2 utility for motor mapping). If the problem persists, contact Adept Customer Service.

[Fatal] Invalid serial I/O configuration (None)

Explanation: During initial startup, eV+ has detected that the configuration of the hardware for serial communications is not valid. An attempt has been made to configure a serial unit that is not installed, or an invalid byte format or baud rate has been requested.

User action: Power down the controller and try starting it again. Make sure that the boot disk you are using is valid for your controller. Use the CONFIG_C utility program to make sure the serial I/O configuration is correct. If the problem persists, contact Adept Application Engineering.

[Fatal] I/O processor failure (-905)

Explanation: The I/O processor on the main CPU board did not start up properly. This processor is critical to the safe operation of the robot. Therefore, if this error occurs, HIGH POWER cannot be enabled.

User action: Power-down and restart your controller. Try a different boot device. Reload your system software. If this problem persists, contact Adept Customer Service for assistance.

[Fatal] Manual mode switch stuck off (-920)

Explanation: During system initialization, a hardware test of the manual mode circuit has found that the key switch is stuck in automatic mode. This error indicates a safety hazard and prevents HIGH POWER from being enabled.

User action: Check any user manual mode switch that may be in use. Verify that the CIP is connected securely. Restart your eV+ system to clear the error and repeat the test. If the problem persists, contact Adept Customer Service.

[Fatal] OVF Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Priv Viol at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Servo code incompatible CPU n (-1102)

Explanation: During initialization, eV+ detected an improper version of servo software on the indicated CPU. eV+ continues to operate, but does not allow high power to be turned on.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service.

[Fatal] Servo dead Mtr n (-1104)

Explanation: The servo process for the indicated motor is not responding to commands from eV+. eV+ continues to operate, but does not allow high power to be turned on.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service.

[Fatal] Servo init failure Board n (-1107)

Explanation: During system initialization the indicated servo interface board cannot be initialized. The problem may be an invalid servo configuration, a missing or improperly configured servo interface board, or a hardware failure.

User action: Power down the controller and restart, making sure you are using the correct system disk. If the problem persists, contact Adept Customer Service.

[Fatal] Servo process dead CPU n (-1101)

Explanation:eV+ failed to detect proper operation of the servo process on the indicated CPU. eV+ continues to operate, but does not allow high power to be turned on.

User action: Power down the controller and restart. Use the CONFIG_C.V2 utility to verify that a servo process is enabled for this CPU. If the problem persists, contact Adept Customer Service.

[Fatal] Spurious Int at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] Stk Overflow at aaaaaa m:n (None)

Explanation: A storage stack within eV+ has overflowed. If n is 1, the error indicates that the eV+ monitor has encountered an expression that has parentheses nested too deeply. Any of the following values for n indicates that the program task shown has attempted to evaluate an expression that is too complex to fit in the stack for that task. The value is a hexadecimal number where ^H1 = monitor task and ^HD = task 0, ^HE = task 1,...^H27 = task 26, and ^H28 = task 27.

User action: If then value is one of those listed above, reduce the complexity of the offending expression. If the value is not one of those listed, an internal problem with eV+ is indicated. In that case, it would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] System clock dead (None)

Explanation: During initial startup, eV+ has failed to detect proper operation of the system clock and timer hardware. eV+ cannot run without the clock operating properly.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

[Fatal] System clock too fast (None)

Explanation: During initial startup, eV+ has detected that the system hardware clock is running too fast. eV+ cannot run without the clock operating properly.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

[Fatal] Uninit Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

[Fatal] ZDIV Trap at aaaaaa m:n (None)

Explanation: An internal problem has occurred with the eV+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart eV+ temporarily by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs. Then power down the controller and restart the system.

File already exists (-500)

Explanation: There is already a disk file or a graphics window with the name supplied to the last storage request.

User action: Reissue the storage request with a different file name, or delete the old file.

File already open (-506)

Explanation: A disk file or graphics window is already open on a logical unit, and another open request has been attempted.

User action: Modify the program to use a different logical unit number for the file or window you want to open, or perform an FCLOSE operation on the file or window currently open on the specified logical unit number before performing the FOPEN operation.

File format error (-512)

Explanation: The requested disk file is not in a format acceptable to eV+ because either it was not created by eV+ or the file has been corrupted.

User action: Use another diskette or reference another file.

File name too long (-570)

Explanation: The file name in an operation was too long.

User action: Use a shorter file name.

File not opened (-513)

Explanation: A program request was made to read or write data from a disk device when no file was open, or an attempt was made to access a graphics window that is not open.

User action: Modify the program to open the file or graphics window before attempting to read or write data.

File or subdirectory name error (-514)

Explanation: The specified file name or subdirectory was not a valid disk file name, the directory path contained invalid syntax, or the directory path was too long.

User action: Retry the operation with a correct file name or subdirectory name. Verify that syntax of the directory path is correct. Verify that any default directory path specified by the DEFAULT command is correct. Verify that the total directory path is not too long when the default is combined with the current file specification.

File too large (-569)

Explanation: The operation caused a file to grow beyond the server's limit.

User action: Close the file, open a new file, and retry the previous operation.

Find: (None)

Explanation: While initiating a string search or replacement operation, the SEE editor is prompting for the string to be found in the program.

User action: Enter the desired search string, or press Return to cancel the request.

First statement must be .PROGRAM (-351)

Explanation: An attempt was made to insert or deposit a program statement above the .PROGRAM statement, which must be the first statement in the program.

User action: Move the cursor to below the .PROGRAM line of the program before attempting to insert or deposit statements.

Font not loaded (-551)

Explanation: The specified font does not exist.

User action: Specify another font (font #1 is always loaded).

Force protect limit exceeded (-624)

Explanation: At least one force-sensor strain gauge reading has exceeded the preset limit, causing a robot panic stop. This may happen due to high forces experienced during an insertion, a crash, or high acceleration.

User action: If a crash occurred, ensure that the work area is cleared. If the limit was exceeded in normal operation, the limit should be increased or Protect mode should be disabled. Enable high power with the pendant and continue operation.

Front panel HIGH POWER lamp failure (-924)

Explanation: HIGH POWER has been disabled because a failure (Open Circuit) in the front panel HIGH POWER lamp has been detected. The lamp is probably burned out. This condition is considered a safety hazard. An E-STOP is not signaled. However, HIGH POWER cannot be enabled until the lamp is replaced.

User action: Replace the HIGH POWER lamp. See the *Adept MV Controller User's Guide*. Re-enable HIGH POWER as desired.

Function already enabled (-422)

Explanation: Certain functions or operations must not be enabled when they are already enabled or active. ALTER mode is an example of such a function.

User action: Avoid reenabling the function or operation.

Graphics processor timeout (-552)

Explanation: The graphics processor (on the system processor) failed to respond to a command from eV+ within five seconds.

User action: Save all your programs and variables on disk and then reboot the system from disk. Contact Adept Customer Service if the problem repeats.

Graphics software checksum error (-558)

Explanation: The code on the graphics board has been corrupted.

User action: Save new or modified programs, restart the controller, and reload the programs. If the problem persists, contact Adept customer service.

Graphics storage area format error (-555)

Explanation: During execution of a FREE command, eV+ has detected that the information in graphic memory may have been corrupted. This may have been caused by a momentary hardware failure or a software error.

User action: Attempt to save as much as possible onto disk. Issue ZERO 1 and ZERO 2 monitor commands to delete graphics data. If the error persists, power down the controller and restart the system.

(HALTED) (8)

Explanation: A HALT instruction has been executed, and thus execution of the current program has terminated.

User action: Any monitor command can be entered, but PROCEED cannot be used to resume program execution.

Hard envelope error Mtr n (-1027)

Explanation: The indicated motor was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system or something impeding the path of the robot. Because this is considered a serious error, high power was turned off.

User action: Turn on high power and try to perform the motion at a slower speed. Make sure that nothing is obstructing the robot's motion. If the error recurs, contact Adept Customer Service.

Hardware not in system (-805)

Explanation: An instruction has attempted to access optional hardware (such as a FORCE board) that is not installed in the system.

User action: Install the needed hardware or remove the instruction that addresses the hardware.

Hard overspeed error Mtr n (-1029)

Explanation: During manual mode, the safety hardware has detected an attempt to move a robot axis at a speed faster than allowed. The motion is terminated and robot power is disabled. This error should never occur if the servos are properly configured.

User action: Verify that the servos for this motor are properly configured. If the problem persists, contact Adept Customer Service.

HIGH POWER button not pressed (-646)

Explanation: You did not press the high power on/off button before the timeout period expired. This message also can result from a faulty cable, Front Panel (FP).

User action: If working from the keyboard, reissue the enable power monitor command and promptly press the high power on/off button when instructed to do so. If working from the MCP, follow the procedure appropriate for enabling high power for the safety category of your system. Promptly press the high power on/off button when instructed to do so. If the timeout period is too short, adjust it by using the ACE controller config tools to change the POWER_ TIMEOUT statement in the eV+ configuration data.

Illegal array index (-404)

Explanation: An attempt has been made to: (1) use a negative value as an array index, (2) use a value greater than 32767 as an array index, (3) specify a simple variable where an array variable is required, (4) omit an array index in a situation where it is required (for example, a 1-dimension array is specified when a 2- or 3-dimension array is required), (5) specify an explicit index in an argument for a eV+ operation that requires a null array, or (6) specify an index to the right of a blank index for a multiple-dimension array.

User action: Correct the line.

Illegal assignment (-403)

Explanation: The assignment operation just attempted was invalid, possibly because it attempted to assign a value to a variable name that is a reserved word or a function.

User action: Reenter the line, using a different variable name if necessary.

Illegal digital signal (-405)

Explanation: A number or bit field specifies a digital signal that is not in one of the allowed ranges or that is not installed. Attempting to set software signal 2032 (brake solenoid) will also give this error.

User action: Correct the signal number and check your digital I/O configuration.

Illegal expression syntax (-458)

Explanation: While decoding a numeric or logical expression, a compound transformation, or a string expression, eV+ has encountered syntax that it does not understand. Possible errors include unmatched parentheses, missing variables, or missing operators.

User action: Retype the line containing the expression, being careful to follow the eV+ syntax rules.

Illegal in debug monitor mode (-359)

Explanation: An operation was attempted that is not accepted in debug monitor mode.

User action: Use a different command, change to debug editor mode, or exit from the program debugger.

Illegal in read-write mode (-365)

Explanation: An editor function was attempted that cannot be performed while accessing a program in read-write mode.

User action: Change to editing the program in read-only mode, or use a different editor command.

Illegal I/O channel number (-518)

Explanation: An internal I/O channel number has been encountered that is invalid. This indicates a eV+ internal software problem.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

Illegal I/O device command (-502)

Explanation: A command to an I/O device was rejected by that device. Certain devices do not accept all commands. For example, random access I/O is illegal to the terminal or to the Kermit device; the GETC function cannot read from a disk file opened for random access. This error may also indicate a hardware problem with the device controller.

User action: Correct the I/O command as required to suit the device. If you continue to have difficulty, contact Adept Application Engineering for assistance.

Illegal I/O redirection specified (-525)

Explanation: An unacceptable I/O redirection has been specified in a DEFAULT monitor command, a disk I/O monitor command (LOAD or STORE_), or in an ATTACH instruction. Either there is a syntax error, or the requested redirection is not allowed for your I/O configuration.

User action: Check the syntax of the offending statement. Make sure that the requested redirection device is allowed on your I/O configuration.

Illegal joint number (-609)

Explanation: A joint number has been specified out of the allowed range.

User action: Correct the joint number.

Illegal memory reference (-418)

Explanation: An operation has attempted to reference an invalid memory address. That is, one that is either out of the allowed range, or that is not in use for any input/output module.

User action: Correct the address or install the missing module.

Illegal monitor command (-300)

Explanation: The name of the command just attempted was not recognized by the system, possibly because it was mistyped or because it was a program instruction and not a command.

User action: Verify the spelling of the command name and enter the command again. Use the DO command to invoke a program instruction from the terminal.

Illegal motion from here (-613)

Explanation: The motion just attempted cannot be performed from the current robot location. For example, NEST can be executed only immediately after a READY instruction; CALIBRATE can be executed only after power-up, LIMP, or NEST; and only CALIBRATE or READY can be executed when the robot is in the nest.

User action: Perform the appropriate operation sequence before retrying the desired motion.

Illegal operation (-423)

Explanation: A program instruction has attempted to perform an operation that is not possible.

User action: Check the instruction executing when the error occurred. Make sure all conditions necessary for its successful completion are met.

Illegal .PROGRAM statement (-467)

Explanation: An attempt has been made to: (1) enter a line other than a .PROGRAM statement as the first line of a program, or (2) enter a .PROGRAM statement that contains a syntax error.

User action: Move below the first line of the program, or reenter the line correctly. (With the eV+ SEE editor, you can press the Undo function key or press Esc+Ctrl+C to cancel the changes you have made to a .PROGRAM line.)

Illegal record length (-528)

Explanation: An FOPEN instruction has specified a record length that is not acceptable. For example, the value is negative or too large, or the record length is zero with random-access

mode specified.

User action: Edit the program to specify a correct record length or specify sequential-access mode.

Illegal use of belt variable (-466)

Explanation: A belt variable has been used in a context where it is not allowed, probably in a compound transformation but not at the left-most position.

User action: Edit the program to use the belt variable correctly.

Illegal user LUN specified (-527)

Explanation: An I/O instruction has specified a logical unit number (LUN) that is not defined in the eV+ system, or cannot be accessed in the manner attempted. (See the description of the ATTACH instruction for a list of the valid logical unit numbers and the devices to which they apply.)

User action: Edit the program to use a logical unit number appropriate for the instruction.

Illegal value (-402)

Explanation: A numeric or expression value that is not in the allowed range was specified within a command or instruction.

User action: Edit the program to use a legal value.

Illegal when command program active (-419)

Explanation: A command program is active and an attempt has been made to execute a command that interferes with operation of the command program. (For example, processing a ZERO command causes the command program to be deleted from the system memory.)

User action: Edit the command program and delete the command causing the error.

Illegal when network enabled (-543)

Explanation: An attempt has been made to perform certain network functions that require that the network be disabled, but the network is enabled.

User action: Disable the network and retry the operation.

Illegal while joints SPIN'ing (-637)

Explanation: An attempt has been made to execute a regular motion instruction while a SPIN trajectory is being executed.

User action: Stop the SPIN trajectory with a SPIN or BRAKE instruction before executing a regular motion instruction.

Illegal while protocol active (-548)

Explanation: This message indicates that you tried to enter passthru mode, or did something unexpected on the serial line configured for use with Kermit while a file was being processed.

User action: Make sure there is no file being accessed by Kermit, and retry the failed operation.

Incompatible safety configuration (-644)

Explanation: The robot and controller do not have the same safety options.

On SmartController systems, this error displays if the Cat 3 option is selected while running a SmartController with an old CIM board. If eV+ detects a 1394 Cat 3 robot when the Cat 3 option is not selected on the SmartController, the message includes the number of the robot associated with the error.

User action: Make sure that the correct robot and controller are being used together. Install (or remove) the appropriate EN954 Safety Category license in the controller.

Initialization error (-505)

Explanation: An I/O device reported an error condition during its initialization. Initialization is performed during power-up, after a reset, and may also be performed after certain nonrecoverable I/O errors occur.

User action: Be sure that the hardware for the I/O device is properly installed. Repeat the failed I/O operation. If the problem persists, contact Adept Customer Service.

Initialization failure Belt n (-1015)

Explanation: The indicated belt encoder monitoring system failed to respond to eV+ during the initialization caused by the DEFBELT instruction.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service. (You can prevent this error from being reported by enabling the DRY.RUN system switch.)

Input block error (-511)

Explanation: A read error has occurred while reading a binary data file from the floppy disk. This indicates that the wrong file was specified or that the data in the file is corrupted.

User action: Try the operation again. If the error recurs, use another diskette.

Input error Try again: (16)

Explanation: The input provided was not consistent with what eV+ expected.

User action: Provide another response.

Interrupted multi-segment motion (-902)

Explanation: Currently, the only multiple-segment motions are MOVEF and MOVESF. For these motion instructions, this error is signaled if the DEPART motion has been initiated, but neither the APPROACH nor the final move to the destination is performed. For example, this situation might occur if an envelope error is detected during the DEPART or the APPROACH motion segments.

User action: Correct the problem that caused the robot motion to terminate prematurely and reexecute or skip the multiple-segment motion.

Invalid argument (-407)

Explanation: An argument for a function, program instruction, or SEE editor command is not in the accepted range.

User action: Verify the range of arguments for the function, program instruction, or editor command being used.

Invalid connection specified (-540)

Explanation: An invalid logical network connection has been specified. For example, a zero connection ID is invalid.

User action: Specify a valid logical connection ID.

Invalid disk format (-520)

Explanation: An attempt has been made to read a disk that is not formatted, or is formatted improperly; or a FORMAT command has been entered that specifies invalid format parameters for the device specified.

User action: If a FORMAT command has been entered, verify the command syntax and retry the command. Otherwise, try a different diskette or reformat the current one. Remember that formatting erases all information on the diskette. If the diskette was created on an IBM PC, be sure it was formatted with one of the formats accepted by the eV+ system.

Invalid error code Belt n (-1010)

Explanation: An unrecognized error was reported by the controller for the indicated conveyor belt.

User action: Attempt the operation again. If the error repeats, report the situation to Adept Application Engineering.

Invalid format specifier (-461)

Explanation: An unrecognized output format specifier was encountered in a TYPE or WRITE instruction, or in an $ENCODE function.

User action: Edit the program to use a valid format specifier.

Invalid hardware configuration (-533)

Explanation: An attempt has been made to access an I/O device in a manner inconsistent with its current configuration. Either the I/O device is not present in the system, or it is configured for some other use. For example, if a serial communication line is configured as a network port, it cannot be accessed as a user serial line.

User action: Make sure the correct device is being accessed. Power down the controller and try starting it again. Make sure the boot disk you are using is valid for your controller. Use the CONFIG_C utility program to make sure the serial I/O configuration is correct. If the problem persists, contact Adept Application Engineering for assistance.

If the error resulted from a disk I/O operation, it indicates that the disk controller hardware is not configured correctly. Adept Customer Service should be contacted in that case.

Invalid in read-only mode (-352)

Explanation: An editor function was attempted that cannot be performed while accessing a program in read-only mode.

User action: Change to editing the program in read-write mode, or use a different editor command.

Invalid network address (-561)

Explanation: This error occurs when a file server has not correctly exported the path being accessed or when an IP network address specified is not of class A, B, or C.

User action: Check the IP addresses used to refer to network nodes.

Invalid network protocol (-541)

Explanation: A message has been received and rejected by a remote node because it does not follow the expected protocol. If the KERMIT device was being accessed, this error indicates that the remote server reported an error or sent a message not understood by the eV+ Kermit driver.

User action: Verify that the network software version on the remote node is compatible with the network software on the local node. DISABLE and ENABLE the affected network nodes and retry the operation. If this error occurs repeatedly, contact Adept Application Engineering for assistance. (For more information about Kermit, see Kermit Communication Protocol.)

Invalid network resource (-560)

Explanation: This error occurs when referring to a node that has not been defined.

User action: Check the node definitions.

Invalid number format (-456)

Explanation: A syntax error was detected while reading a number. For example, an 8 or 9 digit was encountered while reading an octal number.

User action: Reenter the line with a valid number.

Invalid orientation (-619)

Explanation: A motion has been requested to a location that is defined by a transformation with its orientation pointed up instead of down.

User action: Correct the definition of the destination transformation. For example, you may need to correct the base transformation in the compound transformation. (For SCARA robots, the p component of all destination transformations should be approximately 180 degrees.)

Invalid program or variable name (-455)

Explanation: A user-defined name used in a command or instruction was not recognized by eV+.

User action: Verify the name and retype the line.

Invalid qualifier (-476)

Explanation: An invalid qualifier was specified on the last command.

User action: Enter the command again, with a valid qualifier.

Invalid servo error Mtr n (-1001)

Explanation: An unrecognized error was reported for the indicated robot motor.

User action: Attempt the operation again. Contact Adept Customer Service if the error repeats.

Invalid servo initialization data (-625)

Explanation: During eV+ system initialization after booting from disk, servo initialization data in the wrong format was found. This can be caused by using a version of the SPEC utility that is incompatible with the eV+ system.

User action: Make sure your system disk has been configured correctly. Contact Adept Application Engineering for assistance.

Invalid software configuration (-315)

Explanation: During initial startup, eV+ has detected that the system software is not configured properly for the options or hardware present.

User action: Power down the controller and try starting it again. Make sure that the boot disk you are using is valid for your controller. If the problem persists, contact Adept Customer Service for assistance.

Invalid statement label (-463)

Explanation: The program statement label was not an integer from 0 to 65535.

User action: Reenter the line with a valid label.

Invalid steps will be changed to ? lines (None)

Explanation: The AUTO.BAD extended command has been used to change the action to be taken when an invalid line is detected while editing. Subsequently, such a line automatically changed to a bad line with a question mark in column one.

User action: None. This is an informational message.

Invalid when program on stack (-366)

Explanation: An attempt has been made to edit a .PROGRAM or AUTO statement while the program appears on some task execution stack. While a task is on a stack, its subroutine arguments and automatic variable values are kept on the stack. Changes to these statements modify the stack, which is not allowed.

User action: Remove the program from the stack by allowing the task to run until the desired program executes a RETURN instruction, or issue a KILL monitor command to clear the stack. If you are using the SEE program editor, press the Undo key to allow you to continue editing.

Invalid when program task active (-311)

Explanation: An attempt has been made to begin execution of a robot or PC program task when that task is already active.

User action: Abort the currently executing task, or execute the program as a different task, if possible.

I/O communication error (-507)

Explanation: A hardware error has been detected in the I/O interface.

User action: Try your command again. If the problem persists, contact Adept Customer Service.

I/O queue full (-517)

Explanation: Too many I/O requests have been issued to a device too quickly, and there is no more room to queue them.

User action: Retry the operation. If the problem persists, it would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

Is a directory (-568)

Explanation: The caller specified a directory in a nondirectory operation.

User action: Specifying a file that is not a directory, repeat the operation; or perform the correct directory operation.

Joint control of robot not possible (-937)

Explanation: An attempt has been made to perform joint control of a robot for which it is not allowed, like the Adept Quattro robot.

User action: Select a different mode of control. For example, on the pendant you can use WORLD or TOOL mode.

Kinematic solution not found (-936)

Explanation: While evaluating the kinematic solution, the current joint angles of the Adept Quattro robot do not form a consistent set.

User action: Modify the specified destination location (e.g., if a precision point is specified). Press the Brake Release button, and manually move the robot to a different location. If the problem persists, contact Adept Customer Service for assistance.

Line too long (-354)

Explanation: An operation was attempted that would have resulted in accessing a program step that contains too many characters. A single program step can contain at most about 150 characters.

User action: Enter the program step as two or more separate steps.

Location out of range (-610)

Explanation:eV+ has encountered a location that is too far away to represent (possibly within an intermediate computation) or that is beyond the reach of the robot. This probably indicates an error in a location function argument value or in a compound transformation.

User action: Verify that you are using location functions and operations correctly and edit the program as required.

Location too close (-618)

Explanation: An attempt has been made to move the robot to a location that is too close to the robot column. This probably indicates an error in the value of a location function argument or an incorrect compound transformation.

User action: Verify that you are using location functions and operations correctly and edit the program as required.

Macro (Z ends): (None)

Explanation: Definition of a SEE editor macro command has been initiated.

User action: Enter the keystrokes to define the macro and then enter Z to terminate the definition.

Manual brake release (-639)

Explanation: The robot's manual brake-release button is active. It is not possible to enable power when this button is pressed.

User action: Make sure that the manual brake-release button (usually located on the robot) is not active. If the problem persists even though the button is not pressed, call Adept Customer Service.

Manual control pendant failure (-650)

Explanation: A program has attempted to access the pendant when it is disconnected or has failed.

User action: Make sure the pendant is connected properly. If the problem persists, contact Adept Customer Service.

Manual mode cannot be enabled (-932)

Explanation: The safety system has detected a fault condition that prevents manual mode operation of the robot.

User action: Toggle the auto/manual switch. Restart the servos and reboot the controller. Check the safety-related cables. If the problem persists, contact Adept Customer Service.

Manual mode switch 1 off detected by CPU (-917)

Explanation: An E-STOP condition has occurred during manual mode because the CPU has detected on signal channel 1 that the manual mode key switch has been set to automatic

mode. Normally this message is suppressed and error -645 is reported. There may be a hardware problem with the CIP, its cabling, or the AWC.

User action: If safe to do so, toggle the auto/manual key switch and attempt to enable HIGH POWER again. Reseat the plug affixed to the JAWC connector on the CIP and the plug affixed to the CIP connector on the AWC. Verify that the CIP is connected securely. If this error persists, contact Adept Customer Service.

Manual mode switch 2 off detected by CPU (-918)

Explanation: An E-STOP condition has occurred during manual mode because the CPU has detected on signal channel 2 that the manual mode key switch has been set to automatic mode. Normally this message is suppressed and error -645 is reported. There may be a hardware problem with the CIP, its cabling, or, most likely, the AWC.

User action: If safe to do so, toggle the auto/manual key switch and attempt to enable HIGH POWER again. Reseat the plug affixed to the JAWC connector on the CIP and the plug affixed to the CIP connector on the AWC.Verify that the CIP is connected securely. If this error persists, contact Adept Customer Service.

MCP enable switch 1 off detected by CPU (-915)

Explanation: An E-STOP condition has occurred because the CPU has detected on signal channel 1 that the MCP enable switch (formerly called Hold-to-Run) has been released. Normally, this message is suppressed and error -913 is reported. There may be a hardware problem with the CIP or its cabling.

User action: Hold the MCP enable switch and re-enable HIGH POWER as desired. If this error occurs frequently, contact Adept Customer Service.

MCP enable switch 2 off detected by CPU (-916)

Explanation: An E-STOP condition has occurred because the CPU has detected on signal channel 2 that the MCP enable switch (formerly called Hold-to-Run) has been released. Normally, this message is suppressed and error -913 is reported. There may be a hardware problem with the CIP or its cabling.

User action: Hold the MCP enable switch and re-enable HIGH POWER as desired. Reseat the plug affixed to the JAWC connector on the CIP and the plug affixed to the CIP connector on the AWC.

Memory Err at aaaaaa (None)

Explanation: During initialization, eV+ detected a hardware failure at the indicated memory location.

User action: Power down the controller and start it again. If the error persists, contact Adept Customer Service.

Misplaced declaration statement (-471)

Explanation: Upon loading a program or exiting from the program editor, eV+ has encountered an AUTO or LOCAL statement that follows an executable program instruction.

User action: Edit the program to make sure that AUTO and LOCAL statements are preceded only by blank lines, comments, or other AUTO and LOCAL statements.

Missing argument (-454)

Explanation: A valid argument was not found for one or more of the arguments required for the requested command or instruction. That is, the argument was not present at all or an invalid argument was present. A possible cause is the use of a single equal sign (=) for the equality relational operator (==).

User action: Verify the operation syntax and reenter the line.

Missing bracket (-475)

Explanation: In the specification of an array element, a left bracket has been found with no matching right bracket. Either too many left brackets are present or a right bracket has been omitted.

User action: Reenter the line with correctly matching left and right brackets.

Missing parenthesis (-459)

Explanation: An attempt was made to evaluate an expression that did not have correctly matching left and right parentheses.

User action: Correct the expression.

Missing quote mark (-460)

Explanation: A quoted string has been encountered that has no matching quote mark before the end of the line.

User action: Insert a quote mark at the end of the string. Strings may not cross line boundaries.

Motor amplifier fault Mtr n (-1018)

Explanation: The power amplifier for the indicated motor has signaled a fault condition on fault line 1. This fault occurs only for devices controlled by the AdeptMotion Servo system. The interpretation of this fault depends on the particular device being controlled.

User action: Turn high power back on and restart the program. If the error persists, implement procedures appropriate for your AdeptMotion system. If the robot is a standard Adept product, contact Adept Customer Service.

Motor overheating Mtr n (-1016)

Explanation: The indicated motor is overheating.

User action: Reduce the speed, acceleration, and/or deceleration of the robot motions, or introduce delays in the application cycle to give the motor an opportunity to cool.

Motor stalled Mtr n (-1007)

Explanation: The indicated motor has stalled while being driven. This is usually caused by the robot encountering an obstruction.

User action: Turn high power back on and restart the program. Remove the obstruction or modify the program to have the robot follow a different path.

Motor startup failure Mtr n (-1105)

Explanation: During calibration, the indicated motor did not move as expected. The problem may be: (1) the motor is obstructed or up against a limit stop, (2) the load on the robot is too large for calibration, (3) the motor drive hardware is not functioning, or (4) the position encoders are not functioning.

User action: Move the robot away from its limit stops and remove any unusual load. Turn high power back on and try to calibrate again. Contact Adept Customer Service if the error persists.

Must be in debug mode (-360)

Explanation: An editor function was attempted that is accepted only when the program debugger is active.

User action: Use a different editor command or activate the program debugger with the SEE editor DEBUG extended command or the DEBUG monitor command.

Must use CPU #1 (-666)

Explanation: A command or instruction that requires execution on CPU #1 has been attempted on a different CPU.

User: action: Reexecute the command or instruction on CPU #1.

Must use straight-line motion (-611)

Explanation: A joint-controlled motion instruction was attempted while the system was in a mode requiring that only straight-line motions be used. For example, while tracking a conveyor, only straight-line motions can be used.

User action: Change the motion instruction to one that requests a straight-line motion.

Negative overtravel Mtr n* (-1032)

Explanation: The negative hardware overtravel switch for the indicated motor has been tripped. Robot power has been disabled.

User action: Manually move the robot back into range and re-enable power. Check the the negative overtravel switch and its wiring. Use the CONFIG_C utility to adjust the robot soft limits so they are inside the hard limits.

Negative square root (-410)

Explanation: An attempt has been made to calculate the square root of a negative number.

User action: Correct the program as required.

Network closed locally (-535)

Explanation: An attempt has been made to access a DDCMP serial line when the protocol is not active. The protocol was probably stopped because of some other error condition.

User action: Restart the DDCMP protocol.

Network connection closed (101)

Explanation: A client connection has closed on the given logical unit.

User action: None. This is an informational message.

Network connection opened (100)

Explanation: A new client connection has been established on the given logical unit.

User action: None. This is an informational message.

Network connection terminated (-565)

Explanation: This error occurs when input or output operations are attempted on a network connection that has already been terminated.

User action: Reestablish the network connection, and retry the original operation.

Network error Code n (value received)

Explanation: An error code between -255 and -1 (inclusive) has been received from the network. The error code, which does not have meaning to eV+, is being reported to you.

User action: Application dependent. If the indicated code does not having meaning for the current application, verify that the remote computer is sending valid data.

Network node off line (-538)

Explanation: An attempt has been made to send data to a known network node that is off-line. Either the node has been disabled, or it is not connected to the network.

User action: Make sure that the remote node is active and connected to the network. Make sure that the local node is connected to the network.

Network not enabled (-542)

Explanation: An attempt has been made to access a remote network node, or perform certain network functions, when the network is not enabled.

User action: Enable the network and retry the operation.

Network packet partially read (102)

Explanation: A network packet from the UDP driver was too big and was only partially read.

User action: Retry the READ instruction for the UDP device. Change the parameters of the sender to transmit smaller packets.

Network resource already in use (-587)

Explanation: An attempt has been made to use a network address or other network resource that is already in use.

User action: Specify a different address or resource to use.

Network resource name conflict (-564)

Explanation: The name specified matches an existing network resource name.

User action: Choose a different name.

Network restarted remotely (-534)

Explanation:eV+ has received a DDCMP start-up message from the remote system when the protocol was already started. The remote system has probably stopped and restarted its protocol. The local protocol is stopped and all pending I/O requests are completed with this error.

User action: (1) Close and reopen the DDCMP line; (2) check the remote program logic to see why it restarted the protocol.

Network timeout (-562)

Explanation: This error occurs when a network transaction is initiated but no reply is received from the server.

User action: Check network integrity. Make sure the server is up and running. Make sure the correct IP address is being used.

No air pressure (-607)

Explanation:eV+ detected that the air supply to the robot brakes and hand has failed. High power is turned off and cannot be turned on until the air pressure is restored.

User action: Restore the air pressure, turn high power back on, and resume program execution. If the error persists, contact Adept Customer Service.

No data received (-526)

Explanation: An I/O read request without wait has not found any data to return. This is not really an error condition.

User action: Continue polling the I/O device until data is received, or use a read request that waits automatically for data to be received.

No matching connection (-539)

Explanation: A request for a logical network connection has been received and rejected because there is no matching connection on the remote node.

User action: Make sure that the proper logical connection was specified. Make sure that the remote node is operating properly.

Nonexistent file (-501)

Explanation: (1) The requested file is not stored on the disk accessed. Either the name was mistyped or the wrong disk was read.

(2) The requested graphics window title, menu, or scroll bar does not exist.

User action: (1) Verify the file name--use the FDIRECTORY command to display the directory of the disk.

(2) Verify the name of the graphics window element specified.

Nonexistent subdirectory (-545)

Explanation: The subdirectory referenced in a file specification does not exist on the disk that is referenced. Note that the subdirectory may be part of a default directory path set by the DEFAULT monitor command.

User action: Make sure that the subdirectory name was entered correctly. Make sure that the correct disk drive was referenced and that the correct diskette is loaded. Use an FDIRECTORY command to display the directory containing the subdirectory. Make sure that the default directory path is correct.

No other program referenced (-353)

Explanation: A command was issued that attempted to reference a previously edited program, but no other program has been edited during the current editing session.

User action: Use the **New** or **GoTo** function-key command (or the N keyboard command) to change to a new program.

No program specified (-301)

Explanation: No program was specified for an EXECUTE or SEE command or instruction, or DEBUG command, and no previous program is available as a default.

User action: Type the line again, providing a program name.

No robot connected to system (-622)

Explanation: An attempt has been made to attach a robot with a system that does not support control of a robot. (Note that some commands, instructions, and functions implicitly attach the robot.)

User action: Make sure the system has been booted from the correct system disk (for example, use the ID command to display the system identification). Change the program so that it does not attempt to attach the robot.

Not a directory (-567)

Explanation: The caller specified a nondirectory in a directory operation.

User action: Specify a directory in the operation, or use the correct nondirectory operation.

Not attached to logical unit (-516)

Explanation: A program has attempted to perform I/O to a logical unit that it has not attached with an ATTACH instruction. Logical unit 4 allows output without being attached, but all other logical units require attachment for both input and output.

User action: Edit the program to make sure it attaches a logical unit before attempting to use it to perform I/O.

Not configured as accessed (-544)

Explanation: An attempt has been made to access a serial line or other I/O device in a manner for which it is not configured. For example, a Kermit or network line cannot be accessed as a simple serial line.

User action: Verify the proper way to access the serial line for the current configuration. Use the configuration utility program to display the serial line configuration and change it if desired.

Not enough program stack space (-413)

Explanation: An attempt was made to call a subroutine, process a reaction subroutine, or allocate automatic variables when the stack for the program task was too full.

User action: Reorganize the program logic to eliminate one or more nested subroutine calls or reactions; eliminate some of the automatic variables that are allocated by the programs; or use the STACK monitor command to increase the size of the stack for the program task. The program may be restarted with the RETRY command.

Not enough storage area (-411)

Explanation: There is no more space in RAM for programs or variables.

User action: Delete unused programs and variables. If the memory is fragmented because of numerous deletions, it can be consolidated by issuing the commands STORE save_all, ZERO, and LOAD save_all. This writes the memory contents to the disk and reads them back into memory. Note, however, that this procedure does not retain any variables that are not referenced by any program in memory, nor does it retain the values of variables that are defined to be AUTO or LOCAL.

Not found (-356)

Explanation: The search operation was unable to locate the specified string.

User action: Enter a new search string, or consider this an informational message and continue with other operations.

Not owner (-566)

Explanation: The client does not have the correct access identity to perform the requested operation.

User action: Modify the eV+ configuration (using the Adept ACE controller configuration tool) as required to gain access to the server. You may also need to change the access setup on the server itself.

Not yet implemented (-1)

Explanation: This keyword is not implemented in this version of eV+.

User action: Upgrade to a newer version of eV+ or don't use this keyword.

Now reboot the controller to disable Adept Digital Workcell simulation mode.(None)

Explanation: The eV+ monitor command DISABLE ADW has been entered. This message informs you that the Adept controller must be rebooted to actually discontinue Adept Digital

Workcell simulation mode.

User action: If Adept Digital Workcell simulation mode is to be terminated, turn off power to the Adept controller, and turn power back on to reboot the eV+ system. Otherwise, enter the eV+ monitor command ENABLE ADW to cancel the effect of the previous DISABLE command.

Now reboot the controller to enable Adept Digital Workcell simulation mode. (None)

Explanation: The eV+ monitor command ENABLE ADW has been entered. This message informs you that the Adept controller must be rebooted to initiate Adept Digital Workcell simulation mode.

User action: If Adept Digital Workcell simulation mode is to be used, turn off power to the Adept controller, start up the Adept Digital Workcell software on the PC, and turn on power to the Adept controller. Otherwise, enter the eV+ monitor command DISABLE ADW to cancel the effect of the previous ENABLE command.

No zero index Belt n (-1011)

Explanation: The conveyor belt controller did not detect a zero-index mark for the indicated belt.

User action: Make sure the belt encoder is connected properly. If the problem persists, contact Adept Customer Service.

No zero index Mtr n (-1004)

Explanation: The motor controller did not detect a zero-index mark for the indicated joint.

User action: Before you can resume running the program, you must recalibrate the robot. If the problem persists, contact Adept Customer Service.

NVRAM battery failure (-665)

Explanation: The nonvolatile RAM battery backup has failed and the RAM may not hold valid data**.**

User action: Replace the NVRAM battery.

NVRAM data invalid (-661)

Explanation: The nonvolatile RAM has not been initialized or the data has been corrupted.

User action: Power down your controller and restart your system. If the error persists, contact Adept Customer Service.

Obsolete keyword (-2)

Explanation: This keyword is no longer supported.

User action: Don't use this keyword.

Obstacle collision detected (-901)

Explanation: A possible or actual collision has been detected between the robot and a statically defined obstacle. Obstacles may include fixed objects in the workcell as well as structural elements of the robot, such as its base. This error is similar to *Location out of range* in that it is often detected by the kinematic solution programs as the robot is moving.

User action: Move the robot away from the obstacle and continue the motion or modify the executing application program to avoid the obstacle. For application programs, this error may indicate that either the planned end point of the motion collides with an object or that a collision has been detected in the middle of a straight-line motion.

Okay to restart the servos (Y/N)? (12)

Explanation: You have entered the command ENABLE ADW while in Adept Digital Workcell simulation mode. In response to this command, eV+ restarts the servos to account for any robot configuration changes that have made in the Adept Digital Workcell system. eV+ is asking for confirmation before restarting the servos.

User action: If ENABLE ADW was entered on purpose to restart the servos, enter "Y". Otherwise enter "N". See the topic "Communicating with the Controller" in the *Adept Digital Workcell* documentation for information about restarting the servos.

Old value: n, New value: n

Explanation: The specified watchpoint has detected a change in value for the expression being watched. The change occurred because of execution of the program step just before the one indicated.

User action: Enter a PROCEED (Ctrl+P), RETRY, SSTEP (Ctrl+Z), or XSTEP (Ctrl+X) command to resume program execution.[2] Otherwise, enter any other monitor command.

Option not installed (-804)

Explanation: An attempt has been made to use a feature of a eV+ system option that is not present in this robot system.

User action: Power down the controller and try starting it again. Contact Adept Application Engineering if the problem repeats.

Orientation out of range (-935)

Explanation: While evaluating the kinematic solution, the deflection of the moving platform of the Adept Quattro robot was found to be beyond the allowable extreme.

User action: Press the Brake Release button and manually move the robot joints so that the moving platform is nearer to it's "square" shape.

Out of graphics memory (-549)

Explanation: There is no more space in the graphics memory on the system processor for windows, icons, fonts, or other graphics items.

User action: Delete unused graphics items, or reduce the size of windows, to free up graphics memory.

Out of I/O buffer space (-532)

Explanation: An I/O operation cannot be performed because the eV+ system has run out of memory for buffers.

User action: Delete some of the programs or data in the system memory and retry the operation. (Also see *Not enough storage area*.)

Out of network resources (-559)

Explanation: This error applies to many circumstances. Listed below are several possible cases:

1. Too many ports are simultaneously in use for networking; there are no more buffers available for incoming and outgoing packets.

2. Too many drives are being mounted.

3. Too many calls were made simultaneously from separate tasks to a nonfunctional server.

4. Too many node names are being defined.

5. An incoming IP packet was fragmented into too many pieces and eV+ was unable to reassemble it. (This is a highly unlikely occurrence.)

User action: Correct the problem generating the error.

Output record too long (-529)

Explanation: A TYPE, PROMPT, or WRITE instruction has attempted to output a line that is too long. The maximum line length is 512 characters.

User action: Change the program to output less information from each instruction. Remember that you can concatenate the output from separate instructions by using /S to suppress the carriage return and line feed normally done at the end of each TYPE output.

Overtravel Mtr n (-1034)

Explanation: The indicated motor has moved beyond the hardware-limited range of motion.

User action: Manually move the robot back into range and re-enable power. Check the the overtravel switch and its wiring. Use the CONFIG_C utility to adjust the soft limits for the robot so they are inside hard limits.

PANIC command (-633)

Explanation: You have entered a eV+ PANIC monitor command, or a program has executed a PANIC program instruction, which has stopped the current robot motion. High power is still enabled.

User action: To continue the current motion, enter the RETRY monitor command. To continue after the current motion, enter the PROCEED monitor command.

(PAUSED) (9)

Explanation: A PAUSE instruction has been executed, and thus the current program has suspended execution.

User action: Any monitor command can be entered. To continue execution of the program, type **proceed** followed by the task number if it is not 0.

Pendant Not Connected (-657)

Explanation: The pendant is not connected to the XMCP connector or it is not responding.

User Action: Check the connection, cable, and pendant.

Position out of range Jt n (-1002)

Explanation: (1) The requested motion was beyond the software-limited range of motion for the indicated joint; (2) while enabling high power, eV+ detected that the indicated robot joint was outside the software limit.

User action: (1) Modify the program as required to prevent the invalid motion request. (Because the robot did not actually move out of range, you do not need to move the robot before continuing); (2) move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area, and enable power.

Position out of range Mtr n (-1023)

Explanation: (1) The requested motion was beyond the software-limited range of motion for the indicated motor; (2) while enabling high power, eV+ detected that the indicated robot motor was outside the software limit.

User action: (1) Modify the program as required to prevent the invalid motion request (Because the robot did not actually move out of range, you do not need to move the robot before continuing.); (2) move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area, and enable power.

Positive overtravel Mtr n (-1033)

Explanation: The positive hardware overtravel switch for the indicated motor has been tripped. Robot power has been disabled.

User action: Manually move the robot back into range and re-enable power. Check the the positive overtravel switch and its wiring. Use the CONFIG_C utility to adjust the soft limits for the robot so they are inside the hard limits.

Power disabled: Manual/Auto changed (-645)

Explanation:eV+ disables power when the Controller Interface Panel (CIP) switch moves from manual to auto or vice versa.

User action: Use any valid method to enable high power.

Power failure detected (-667)

Explanation: Indicates that a controller AC power-fail condition has been detected. If battery backup is installed, this error is reported (when power is restored) by any I/O operations that were canceled due to the power failure. This error code may be trapped by a program using the REACTE instruction in order to provide some level of automatic power failure response.

User action: You may need to restart or repeat any operations that were interrupted by the controller AC power failure. Some reinitialization of the system may be required: for example, any robot(s) connected to the controller must be recalibrated after a controller power failure.

Power failure detected by robot (-632)

Explanation: Indicates that a robot amplifier has detected an under-or over-voltage of its internal DC Bus. This may be due to AC power supply out of spec of 200V-240V or a motion that is too hard or too fast for the payload of the robot.

User action: Verify the voltage on the AC input line. Lower the speed, acceleration or deceleration of the robot. Reduce the payload if possible.

Press HIGH POWER button to enable power (57)

Explanation: The high power on/off button on the front panel must be pressed to complete the process of enabling high power.

User action: When the high power on/off button on the Controller Interface Panel (CIP) blinks, promptly press the button to complete the two-step process of enabling high power. (You must press the button within the time period specified in the eV+ configuration data.)

Press HIGH POWER button when blinking (60)

Explanation: When enabling power for Cat3 systems which are in manual mode, the HIGH POWER ON/OFF button on the front panel must be pressed after the Hold-to-run button has been released and held. The light on the HIGH POWER ON/OFF button will blink when it is time to press it. You must press the button within the time period specified in the eV+ configuration data (which by default is ten seconds).

User action: When the HIGH POWER ON/OFF button on the VFP blinks, promptly press the button to complete the two-step process of enabling high power.

Processor crash CPU = n (None)

Explanation:eV+ has detected that the specified CPU within the controller has entered a fatal error state. A crash message from that processor is displayed immediately following. A software error or hardware problem with that processor is likely.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred. You should store the programs that are in memory, power down the controller, and start it again. (If the processor ID shown is 1, you can restart eV+ by pressing Ctrl+G. The robot servos do not function, but you can STORE the programs in memory.) If the problem persists, contact Adept Customer Service.

Program already exists (-309)

Explanation: An attempt has been made to LOAD a program that already exists, or to COPY or RENAME a program to a name that is already in use.

User action: Delete the conflicting program or use a different name.

Program argument mismatch (-408)

Explanation: The arguments in a CALL, CALLS, or EXECUTE instruction do not match the arguments in the program being referenced because they are of different types.

User action: Modify the CALL, CALLS, or EXECUTE instruction, or the .PROGRAM statement of the referenced program, so that the argument types match. If arguments are omitted in

the CALL, CALLS, or EXECUTE instruction, make sure the appropriate commas are included to position the arguments that are present.

Program completed (3)

Explanation: The program has been executed the number of times specified in the last EXECUTE command or instruction.

User action: Any monitor command can be entered, except that PROCEED cannot be used to resume program execution.

Program HOLD (15)

Explanation: The RUN/HOLD button on the pendant has been pressed while a robot program was executing, and it is now suspended.

User action: Any monitor command can be entered. To continue execution of the program, type **proceed** or **retry**, or press the PROGRAM START button on the controller. (The RUN/HOLD button can be held down to temporarily resume execution of the program if the front-panel keyswitch is in the MANUAL position.)

Program interlocked (-308)

Explanation: An attempt has been made to access a program that is already in use by some eV+ process. For example, you have attempted to delete or edit a program that is being executed, or execute a program that is being edited.

User action: Abort the program or exit the editor as appropriate and retry the operation. You can use the SEE editor in read-only mode to look at programs that are interlocked from read-write access.

Program name? (None)

Explanation: A SEE editor command to change to a different program has been entered.

User action: Enter the name of the new program to be edited, or press Return to cancel the request.

Program not executable (-307)

Explanation: Because of program errors detected during loading or upon exiting from the editor, this program cannot be executed.

User action: Edit the program to remove any errors.

Program not on top of stack (-421)

Explanation: A DO context specification has referenced an automatic variable or a subroutine argument in a program that is not on the top of the stack for the specified task.

User action: Reenter the DO command and specify the correct program context or eliminate references to automatic variables and subroutine arguments. Use the STATUS command to determine which program is on the top of the stack.

Program program_name doesn't exist. Create it (Y/N)? (None)

Explanation: An attempt has been made to use the SEE editor to access a program that does not currently exist.

User action: Enter a Y to have the program created. Any other input, including just pressing Return, cancels the edit request.

Program task # stopped at program_name, step step_number date time (4)

Explanation: Execution of the program task indicated by # has terminated for the reason indicated in the message that preceded this message. The step number displayed corresponds to the next NEXT program step that is executed (for example, if PROCEED were entered). The current date and time are displayed if the system date and time have been set.

User action: None. This is only an informational message.

Program task not active (-318)

Explanation: An attempt was made to abort a task that was not active.

User action: None required if the correct task number was specified. Otherwise, use the STATUS command to determine which task number should have been used.

Program task not in use (-319)

Explanation: A program task cannot be accessed because it has never been used. (Such program tasks do not use any system memory and do not appear in the STATUS display.)

User action: None.

Protected program (53)

Explanation: An attempt has been made to list a program that is protected from user access.

User action: None.

Protection error (-530)

Explanation: An I/O operation cannot be performed because (1) it attempted to write to a disk that is write protected, or (2) you do not have the proper access status.

User action: Check the diskette to make sure the write-protect tab is in the correct position. Use an FDIRECTORY command to display the disk directory. If the file has protected (P) or read-only (R) protection, you cannot access it in the way attempted.

Push, release, hold, MCP enable switch (58)

Explanation: You are attempting to enable HIGH POWER while in manual mode. This operation requires that the MCP enable switch be tested. HIGH POWER cannot be enabled unless you toggle the switch to test it.

User action: Push, then release, then hold the enable switch within a limited time. Alternately, you can change from manual mode to automatic mode.

Recursive macros illegal (-357)

Explanation: An attempt was made to execute a macro recursively. That is, the macro contained a command character sequence that (directly or indirectly) restarted execution of the macro.

User action: Change the macro definitions as necessary to make sure neither macro invokes itself. You can have the U macro invoke the Y macro, or vice versa (but not both).

*Release then press Hold-to-run button (58)

Explanation: When enabling power for Cat3 systems which are in manual mode, the Hold-to-run button in the handle of the MCP must be released then pressed and held. The release time must be between two and ten seconds.

User action: Promptly release then press the button when this message is displayed. Make sure that you press the Hold-to-run switch and not the Run/Hold button on the top of the MCP.

Remote digital input failed Block n (-1112)

Explanation: If displayed during startup, eV+ has failed to detect the specified remote digital input block on the servo network. This is caused by one of the following conditions: the required node is not connected to the network; the eV+ I/O configuration is incorrect; or the configuration of the servo nodes is incorrect. If encountered during normal operation, the remote digital input block has stopped supplying data. The servo network or remote network node may have failed.

User action: If this error occurs during startup, verify that all required network nodes are connected. Use the SRV.NET monitor command to verify that the network nodes contain the desired input blocks. Verify that the eV+ I/O configuration matches the network node digital input configuration. If this error occurs during normal operation, reinitialize your servo network or reboot your controller. If the problem persists, contact Adept Customer Service.

Remote digital output failed Block n (-1113)

Explanation: During startup, eV+ has failed to detect the specified remote digital output block on the servo network. This is caused by one of the following conditions: the required node is not connected to the network; the eV+ I/O configuration is incorrect; or the configuration of the servo nodes is incorrect.

User action: Verify that all required network nodes are connected. Use the SRV.NET monitor command to verify that the network nodes contain the desired output blocks. Verify that the eV+ I/O configuration matches the network node digital output configuration.

Remote has not exported network resource (-563)

Explanation: The server has not exported the designated path for use by clients.

User action: Check the server setup, and check the path that the eV+ system uses.

Reserved word illegal (-457)

Explanation: An attempt has been made to use a eV+ reserved word for a variable name.

User action: Use a different name for the variable. You can, for example, append a prefix or suffix to the attempted name.

Return manual control pendant to background display (^C to exit) (None)

Explanation: The pendant display must be in background mode for the operation you have selected.

User action: Press the DONE button on the pendant one or more times to exit the current function.

Robot already attached to program (-602)

Explanation: A program has executed more than one ATTACH instruction for the robot, without executing a DETACH in between. Or an attempt has been made to SELECT another robot when one is already attached. The robot is still attached even after this error occurs.

User action: Check the program logic-remove redundant ATTACH instructions, or DETACH the current robot before attempting to SELECT another robot.

Robot already under manual control (-938)

Explanation: An attempt has been made to control the robot from the Adept ACE virtual pendant, or a JOG program instruction or Monitor command has been issued, while the robot is already being controlled from the pendant.

User action: COMP mode from the pendant.

Robot configuration option or model illegal (-930)

Explanation: During the booting sequence, a selected robot kinematic module detected a specified optional configuration mode or model number that is not supported in either this version of eV+ or by this kinematic module. Often, this occurs if you have enabled a feature that is only available in enhanced kinematic module and you have loaded the standard kinematic module.

User action: Verify that you have the required version of eV+ and that the kinematic module supports the robot model and configuration options that you have selected. If you do not have a license for the required kinematic module, either turn off the option or purchase the required license.

Robot interlocked (-621)

Explanation: (1) An attempt has been made to access a robot or external device that is already being used by a different program task or by the system monitor; (2) an attempt has been made to calibrate the robot with the controller in manual mode, which is not allowed for safety reasons.

User action: (1) Review the program logic and make sure the robot or device is being controlled by only one program task; (2) Move CIP key switch to the automatic mode position or set the user manual mode signals appropriately.

Robot module not enabled (-900)

Explanation: The indicated robot module is present in memory, but it was not enabled for use due to an error (which is reported by a separate message).

User action: Use the CONFIG_C and/or SPEC utilities to correct the module configuration.

Robot module not loaded ID: n (-628)

Explanation: This error occurs only during startup when a robot module has been configured using the CONFIG_C utility, but the robot module is not present in memory.

User action: Use the CONFIG_C utility to add the robot module to the boot disk before rebooting.

Robot not attached to this program (-601)

Explanation: An attempt has been made to execute a robot-control command or instruction in one of the following invalid situations:

(1) The system is not configured to control a robot. (2) There is no robot connected to the system. (3) The robot is attached to a different program task.

User action: (1) Make sure the system is booted from the proper system disk, or remove the robot-control instruction.

(2) Connect the robot or enable the DRY.RUN system switch.

(3) Modify the program logic as required to ensure that only one program task is controlling the robot at any given time.

Robot not calibrated (-605)

Explanation: An attempt has been made to execute a robot-control program when the robot is not calibrated. No motion is allowed until the robot is calibrated.

User action: If you want to use the robot, issue a CALIBRATE command or have your program execute a CALIBRATE instruction. Or enable the DRY.RUN switch to allow program execution without using the robot.

Robot overheated (-606)

Explanation: (1) Robot joint 1 has been moved into the hardware brake track area, which causes high power to be turned off and prevents the robot from moving.

(2) The robot base has become overheated.

User action: (1) Push the brake release button at the robot base and move the joints back into the normal working range. Turn on high power and continue program execution.

(2) Check the fan filter on the robot base, and check the ambient temperature of the robot. Allow the robot to cool down, turn on high power, and continue program execution.

Robot power off (-604)

Explanation: The requested operation cannot be performed because HIGH POWER is off.

User action: Enable power and retry the operation.

Robot power off requested (-906)

Explanation: HIGH POWER has been turned off because of a program or user request, such as issuing a DISABLE POWER command.

User action: None required.

Robot power on (-627)

Explanation: An attempt has been made to perform an action that requires high power to be off.

User action: DISABLE POWER and reexecute the action.

Robot power system failure Code n (-1115)

Explanation: The servo interface has detected a fault in the robot power system. The code number provides more detailed

information about the fault. Consult your amplifier reference manual to interpret the code values.

User action: Depends on the particular code value. In general, check the servo and amplifier cabling. Power-down and restart your controller and servo nodes. If the problem persists, contact Adept Customer Service.

RSC bad packet format (-655)

Explanation:eV+ has received an incorrect data packet from the robot signature card, during the initial calibration data load.

User action: None unless the calibration load fails. If the problem persists, contact Adept Customer Service.

RSC calibration load failure (-656)

Explanation:eV+ cannot load calibration data from the robot signature card (RSC).

User action: Power down the controller and make sure the robot cables are correctly and securely connected. If the problem persists, contact Adept Customer Service.

RSC communications failure (-651)

Explanation:eV+ has lost communications with the robot signature card (RSC). Either a hardware problem has occurred or the robot is being operated in an environment with excessive electrical noise.

User action: Check the connections of the robot cables. Turn high power back on, calibrate the robot, and resume program execution. If the problem persists, contact Adept Customer Service.

RSC hardware failure (-669)

Explanation: The RSC has reported an internal failure. Because RSC failures almost always cause the RSC to stop communicating altogether (rendering it incapable of reporting the failure), this error message may be due to some other cause, such as electrical noise at the RSC or within or around the arm signal cable.

User action: If the problem persists, contact Adept Customer Service.

RSC module ID doesn't match robot (-676)

Explanation: The eV+ configuration data contains an explicit ID specification for a robot module (for example, 6 for the Adept 550 robot), and the robot RSC does not contain that ID number.

User action: Make sure that the correct type of robot is being used. Use the CONFIG_C utility to change the module ID to -1 in the eV+ configuration data. Correct the module ID in the RSC.

RSC power failure (-670)

Explanation: The RSC has reported that its power is failing. Because a power failure on the RSC almost always causes it to stop communicating altogether (rendering it incapable of reporting the failure), this error message may be due to some other cause, such as electrical noise at the RSC or within or around the arm signal cable.

It is possible that the power lines to the RSC have an intermittent connection somewhere.

On FireWire Robots this error indicates that a robot amplifier has detected an under- or over-voltage of its internal DC Bus. This may be due to AC power supply out of spec of 200-240V or a motion that is too hard or too fast for the payload of the robot.

User action: If the problem persists, contact Adept Customer Service.

RSC reset (-652)

Explanation:eV+ has detected that the robot signature card (RSC) has lost power temporarily, but is now functioning.

User action: Turn high power back on and resume program execution. If the problem persists, check the cabling to the robot. Contact Adept Customer Service if no solution can be found.

RSC time-out (-653)

Explanation:eV+ has not received a response from the robot signature card (RSC) when expected, during the initial calibration data load. The RSC or its cabling is probably faulty.

User action: Power down the controller and check the cables to the robot. If the problem persists, contact Adept Customer Service.

RSC transmission garbled (-654)

Explanation:eV+ has received an invalid transmission from the robot signature card (RSC). Either a hardware problem has occurred or the robot is being operated in an environment with excessive electrical noise.

User action: None unless the calibration load fails or RSC communications fail. If the problem persists, contact Adept Customer Service.

Safety speed limit error Chn n (-1035)

Explanation: During system startup or a servo reset, the servo self-test has determined that the safety speed limit hardware is not functioning properly. Robot power cannot be

enabled.

User action: Power-down and restart your controller and servo nodes. If the problem persists, contact Adept Customer Service.

Safety system fault Code n (-1109)

> **NOTE:** If you are using an AIB robot, such as the Cobra s600/s800, refer to the document Adept Status Code Summary for Embedded Products.

Because the status codes are related to hardware, refer to your robot hardware documentation as your primary source of information for correct hardware and safety system setup. You should also re-commission the robot. (For details, see the hardware documentation for your robot.) If it does not resolve the problem, contact Adept Customer Service.

Safety system init failure Code n (-1108)

> **NOTE:** See also details of messages codes in error -1109.

Because the status codes are related to hardware, refer to your robot hardware documentation as your primary source of information for correct hardware and safety system setup. You should also re-commission the robot. (For details, see the hardware documentation for your robot.) If it does not resolve the problem, contact Adept Customer Service.

Safety system not commissioned (-648)

Explanation: A system with the EN954 Safety Category 3 option (pre-2012)—the Manual Mode Safety Package (MMSP)—has not been successfully commissioned with the SAFE_UTL utility program.

OR

A system with the PL=d Safety (according to ISO 13849) has not been successfully commissioned with Adept ACE Safety Commissioning Utility or has been decommissioned by the internal self-checking firmware.

User action: Test the safety with the SAFE_UTL utility program or Adept ACE Safety Commissioning Utility before enabling power for the first time. You should then rerun the SAFE_UTL utility program or Adept ACE every three months to re-commission the robot. If you have connected the robot to a different controller or replaced the controller or the SIO module, repeat the test. (For details, see the documentation for the SAFE_UTL program or the *Adept ACE User's Guide*, and the hardware documentation for your robot.)

Searching for string (exact case) (None)

Explanation: The SEE editor command 0' has been entered. The editor is prepared to search for the string indicated, in the search mode indicated.

User action: This is an informational message. You can use the **Repeat** command to perform the indicated search, or you can use **Find** (or **Change**) to initiate a new search (or replacement) operation. The EXACT extended command controls the setting of the search mode.

Searching for string (ignoring case) (None)

Explanation: The SEE editor command 0' has been entered. The editor is prepared to search for the string indicated, in the search mode indicated.

User action: This is an informational message. You can use the **Repeat** command to perform the indicated search; or you can use **Find** (or **Change**) to initiate a new search (or replacement) operation. The EXACT extended command controls the setting of the search mode.

Servo board E-Stop fuse open (-673)

Explanation: Your servo board has a fused E-STOP circuit, and the system has detected an open circuit at that location.

User action: Refer to your hardware documentation, consult with Adept Customer Service as needed for details about types and locations of fuses, and replace the fuse.

Servo board 12v fuse open (-671)

Explanation: Your servo board has a fused 12-volt bus, and the system has detected an open circuit at that location.

User action: Refer to your hardware documentation, and replace the fuse.

Servo board solenoid fuse open (-672)

Explanation: Your servo board has a fused robot solenoid control line, and the system has detected an open circuit at that location.

User action: Refer to your hardware documentation, and replace the fuse.

Servo hardware init failure (-931)

Explanation: During system startup or after a servo reset, the servo software had detected that it could not initialize its hardware. This is a generic error when more specific error information is not available.

User action: Restart the servos and reboot the controller. Check the safety-related cables. If the problem persists, contact Adept Customer Service.

Servo node not downloaded (-680)

Explanation:eV+ has attempted to use a servo node that does not contain properly downloaded software. Servo node software is only loaded at eV+ startup time. One of the following has happened: 1. You are using a servo node that is not supported by the eV+ system software; 2. You have plugged in a new servo node after eV+ was started.

User action: Reboot eV+ and try again. Make sure your software is compatible with your 1394 nodes. Update the 1394 firmware.

Servo protocol incompatible(-677)

Explanation: During startup, eV+ has detected a remote node on the servo network that is incompatible with the current eV+ system because incompatible versions of software are being used. The servo network does not operate.

User action: Use the SRV.NET monitor command to determine which network nodes are causing this error. Verify that the proper eV+ software is being used. Verify that the servo nodes were configured using the proper utility program version.

Servo task overloaded (-674)

Explanation: A servo interrupt task has used up all the execution time. The detection algorithm reports an error when the servo interrupt task completely occupies 10 or more time slices per second of real time. The robot went to a fatal error state when this error occurred, and the servo interrupt task stopped running.

User action: Change one or more of the following: (1) move servo tasks off CPU #1 to allow more time for trajectory generation, (2) upgrade the system processor to increase the throughput, or (3) reduce the number of robots or axes that you are operating.

Set for CASE DEPENDENT searches (None)

Explanation: The EXACT extended command has been used to change the method by which character case is considered during string searches. The message indicates how case is considered in subsequent searches (for the current or future search-for strings).

User action: None. This is an informational message.

Set for CASE INDEPENDENT searches (None)

Explanation: The EXACT extended command has been used to change the method by which character case is considered during string searches. The message indicates how case is considered in subsequent searches (for the current or future search-for strings).

User action: None. This is an informational message.

Skew envelope error Mtr n (-1022)

Explanation: The two motors associated with a split robot axis were not tracking each other with sufficient accuracy.

User action: Make sure nothing is obstructing the robot motion. Turn on high power and try to perform the motion at a slower speed. If necessary, use the SPEC utility to increase the maximum skew error.

Soft envelope error Mtr n (-1006)

Explanation: The indicated motor was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system or something impeding the path of the robot. Because this was not considered a serious error, a controlled motion stop occurred and high power remains on.

User action: Try to perform the motion at a slower speed. Make sure nothing is obstructing the robot's motion.

Soft overspeed error Mtr n (-1028)

Explanation: During manual mode, the servos have detected an attempt to move a robot axis at a speed faster than allowed. The motion is terminated but robot power remains enabled.

User action: Move the robot at a slower speed. If you are near a mechanical singularity, move the robot using *joint* mode instead of *world* or *tool* mode.

Software checksum error (-316)

Explanation: During processing of a FREE command the eV+ system has detected a checksum error in the system memory. This indicates a problem with the system software or hardware. (Note, however, that a checksum error is introduced if any patches are made to the system software after the system is loaded from disk and started up.) The following codes are appended to the message indicating where the error occurred: Os, operating system; eV+, eV+ interpreter or trajectory generator; Vi, vision software; Sv, servo software.

User action: Report to Adept Application Engineering the error and information about any possible contributing circumstances. You can continue to use the system, but you should keep in mind the possibility of a problem with the hardware.

Software incompatible Code n (-1026)

Explanation: The servo code has detected an incompatibility between the servo code and calibration software.

User action: Make sure that you are using the calibration software (in the \CALIB\ directory) that you received with the eV+ system you are using. If you are using the correct software, note the code number, and call Adept Customer Service.

Speed pot or STEP not pressed (-620)

Explanation: While the controller was in manual control mode, a eV+ program tried to initiate a robot motion, but you failed to press the step button and speed bar on the MCP.

User action: When a eV+ program is about to initiate robot motions, press the step button and speed bar on the MCP. To continue the motion once it has started, you can release the step button but must continue to press the speed bar. Set the controller to automatic mode.

SPIN motion not permitted (-638)

Explanation: Either a SPIN instruction has attempted to move a joint that has not been configured with the continuous-rotation capability or the robot is currently tracking a belt or moving under control of an ALTER instruction.

User action: Configure the joint with continuous-rotation capability, or complete the belt tracking or ALTER instruction before attempting to execute the SPIN instruction.

Step syntax MUST be valid (None)

Explanation: The SEE editor's AUTO.BAD extended command has been used to change the action to be taken when an invalid line is detected while editing. Subsequently, the editor requires that such a line be corrected before you are able to perform any operation that moves the cursor off the bad line.

User action: None. This is an informational message.

Stop-on-force triggered (-623)

Explanation: A force-sensor Guarded Mode trip occurred when the robot was not under program control.

User action: High power must be re-enabled before robot motion may continue. If the trip was not desired, make sure that Guarded Mode is disabled before the program relinquishes control of the robot to the manual control pendant.

Stopped due to servoing error (-600)

Explanation: Program execution has stopped because of one or more servo errors.

User action: Correct the source of the reported servo errors, referring to your system hardware manual as required.

Storage area format error (-305)

Explanation: During execution of a FREE command, eV+ has detected that programs or data in RAM may have been corrupted. This may have been caused by a momentary hardware failure or a software error.

User action: Attempt to save as much as possible onto the disk. Then enter a ZERO command or power down the controller and restart the system.

Straight-line motion can't alter configuration (-612)

Explanation: A change in configuration was requested during a straight-line motion. This is not allowed.

User action: Delete the configuration change request, or use a joint-interpolated motion instruction.

String too short (-417)

Explanation: A program instruction or command expected a string argument with a certain minimum length and received one that was too short.

User action: Review the syntax for the program instruction and edit the program to pass a string of the correct length.

String variable overflow (-416)

Explanation: An attempt has been made to create a string value that is greater than the maximum string length of 128 characters.

User action: Edit the program to generate strings of the proper length.

Subdirectory in use (-547)

Explanation: An attempt has been made to delete a subdirectory that still contains files or that is being referenced by another operation (for example, an FDIRECTORY command).

User action: Make sure that all the files within the subdirectory have been deleted. Make sure that no other program tasks are referencing the subdirectory. Retry the delete operation.

Subdirectory list too long (-546)

Explanation: A directory path contains too many subdirectories, or the directory path is too long to be processed. The path is a combination of subdirectories in the file specification and the default directory path set by the DEFAULT monitor command. Directory paths are limited to a total of 16 subdirectories and 80 characters (including any portion of the directory path specified by the current default path).

User action: Specify a shorter directory path in the file specification or in the DEFAULT command. If you are accessing a foreign disk that contains more than 16 nested subdirectories, you cannot read the files in subdirectories nested deeper than 16 levels. In this case, you must use the system that created the disk to copy the files to a directory that is nested less deeply.

Switch can't be enabled (-314)

Explanation: An ENABLE command for a certain switch has been rejected because of some error condition. For example, ENABLE POWER fails if the system is in FATAL ERROR state.

User action: Review the description for the switch you are trying to enable, correct the error condition, and try again.

SYSFAIL detected by CPU (-629)

Explanation: A board on the VMEbus has encountered a severe error and asserted the SYSFAIL signal which turns off HIGH POWER. The watchdog timers on the CPU boards assert this signal and light the SF LED if severe software errors occur.

User action: Check the SYSFAIL LEDs on the front edge of the boards. The board which has failed should light its LED. Restart the system. Verify proper seating of the system boards and correct device connections to the boards. Test the system with as many boards removed as possible, adding boards back in until the problem board is identified. If the problem persists, contact Adept Customer Service.

SYSFAIL detected by robot (-642)

Explanation: The motion interface board has detected a SYSFAIL signal on the VMEbus and has asserted the backplane E-STOP signal. This error is normally superseded by other errors and not seen.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number of the task that received the error. If no additional information is available, call Adept Customer Service.

Task = (None)

Explanation: The SEE editor DEBUG extended command has been used to initiate a program debugging session for the current program. The debugger needs to know which program task you want to use when executing the program.

User action: Enter the desired task number, or press Return to access the same task used for the last debugging session.

Template already defined (-748)

Explanation: When defining a new correlation template with the program instruction VTRAIN.MODEL, the number of an existing template was given.

User action: Delete the existing template if it is no longer needed, or use a different number in the VTRAIN.MODEL instruction.

Template of uniform intensity (-746)

Explanation: When defining a correlation template with the VTRAIN.MODEL program instruction, the area of the image within the given template bounds has uniform intensity. Image templates must have some variation in brightness. (That is, there must be some features in the template to correlate with later.)

User action: Check the position of the template in the image and make sure it is in the desired place. Also, view the grayscale image in the current frame to make sure it is valid. (For example, maybe a strobe light did not fire, or the lens cap is still on the camera.)

Template not defined (-747)

Explanation: The correlation template referenced in a VCORRELATE, VDELETE, VSHOW.MODEL, or VSTORE operation does not exist.

User action: Verify the correlation number supplied to the operation. Use the Models pull-down menu in the vision window (or the VSHOW.MODEL program instruction) to get a list of the templates currently defined in the vision system.

Time-out enabling amplifier Mtr n (-1009)

Explanation: The power amplifier for the indicated motor has signaled a fault condition. A momentary power failure or a hardware error may have occurred.

User action: Turn high power back on and restart the program. If the error persists, contact Adept Customer Service.

Timeout enabling power (-675)

Explanation: High power did **not** enable within the allowed amount of time, and the servos reported no other error during the timeout period.

User action: For non-Adept robots, use the SPEC utility to increase the value of the high power time-out.

For Adept robots, double-check your installation (cabling, AC power line voltages, circuit breakers, amplifier retaining screws, cables, and contactors). For information about the correct configuration for installation, refer to your Robot Instruction Handbook. Make sure that the amplifier chassis is properly connected to a power source and is turned on. Try again. If the problem persists, contact Adept Customer Service.

Timeout: MCP enable switch not toggled (-649)

Explanation:eV+ did not enable high power because you failed to properly toggle the MCP enable switch on the manual control pendant.

User action: Do one or more of the following: (1) when toggling the MCP enable switch, release it for a minimum of two seconds and a maximum of ten seconds, and then press it back in; and (2) make sure that you are pressing the MCP enable switch and not the run/hold button by mistake.

Time-out nulling errors Mtr n (-1003)

Explanation: The indicated motor took too long to complete the last motion, possibly because the robot is blocked and cannot reach its destination.

User action: Retry the motion after making any necessary program changes. If this error occurs repeatedly, contact Adept Application Engineering for assistance.

Too many arguments (-553)

Explanation: Too many arguments were specified for the last command or instruction.

User action: Reenter the command or instruction but with the correct number of arguments.

Too many array indices (-474)

Explanation: The specification of an array element contains more than three indexes.

User action: Reenter the line with the correct number of indexes.

Too many closeable windows (-554)

Explanation: The names of too many graphics windows have been specified to appear in the pull-down under the Adept logo in the status line at the top of the screen.

User action: Specify all subsequent windows as /NOCLOSEABLE, or delete some existing windows that appear in this pull-down.

Too many network errors (-536)

Explanation: (1) The number of errors detected by the DDCMP protocol has exceeded the maximum allowed. The local protocol is stopped, and all pending I/O requests are completed with this error.

(2) The eV+ Kermit driver experienced more errors than permitted by the KERMIT.RETRY parameter.

User action: (1) Use the NET monitor command to determine the type of errors that have occurred. Check for noise on the communication line, errors in the remote DDCMP implementation, or program logic that sends messages faster than they can be processed. Use the appropriate FCMND instruction to increase the maximum number of errors.

(2) Set the KERMIT.RETRY parameter to a larger value, increase the retry threshold on the remote server, restart the Kermit session, and retry the operation that failed.

Too many vision requests pending (-703)

Explanation: A program has issued too many VLOCATE commands before the first ones have completed.

User action: Edit the program to wait for pending VLOCATE requests to complete before issuing more.

Too many windows (-550)

Explanation: An attempt was made to create a graphics window when the maximum number of windows were already defined. (The eV+ system uses two windows for the screen and the top status line. Every title bar, menu bar, and scroll bar is a separate window. The pull-down window is always allocated even if it is not visible. Systems with AdeptVision always have the vision-training window allocated.)

User action: Where possible, change your window definitions to omit menu bars and scroll bars. If necessary, use the utility program CONFIG_C to increase the number of window buffers.

Trajectory clock overrun (-636)

Explanation: One of these three conditions has occurred: (1) the time for a new trajectory point has arrived, but the internal trajectory task has not finished computing the previous point; (2) the servos did not receive trajectory data at the expected time because the trajectory task took too long to compute and write out the data; or (3) the trajectory interval is equal to or less than the servo interval.

User action: Perform one or more of the following: (1) if the trajectory cycle time is less than 16 msec, change it to the next longer time; (2) move servo tasks off CPU #1 to allow more time for trajectory generation; (3) upgrade the system processor to increase the throughput; (4) reduce the number of robots or axes that you are operating; or (5) if the trajectory cycle time is set to 2 msec, make sure the servo interval is 1 msec.

Undefined program or variable name (-406)

Explanation: The program or variable, referenced in a command or program step, does not exist-possibly because the name was mistyped.

User action: If the correct name was entered, create the program or variable using one of the eV+ editors or the appropriate eV+ monitor commands, or by loading from a disk file.

Undefined value (-401)

Explanation: (1) A variable has been referenced that has not been assigned a value.

(2) Using the SEE editor, an attempt has been made to use a macro, return to a memorized cursor position, or perform a repeat string search or change without first performing the appropriate initialization sequence.

User action: (1) Assign the variable a value or correct its name.

(2) Define the macro, record a cursor position, or enter the desired search/replacement string(s).

Undefined value in this context (-420)

Explanation: An automatic variable or subroutine argument value appears in a monitor command, but the specified program is not on the execution stack for the specified program task. Automatic variables and subroutine arguments have values only when the program that defines them is on a stack.

User action: Change the monitor command to not reference the variables. Make sure that the program is on the expected execution stack. You can place a PAUSE instruction or breakpoint in the program to stop it while it is on the execution stack.

Unexpected end of file (-504)

Explanation: (1) If a file was being loaded from the disk, the end of the file was encountered unexpectedly.

(2) If a program is reading a file, this error code merely indicates that the end of the file has been reached and should not be interpreted as a real error.

(3) This message results if a CTRL+Z is pressed in response to a program PROMPT.

(4) A break condition was detected on a serial line.

User action: (1) Try again to read the file.

(2) Close the file and continue program execution.

(3) Treat the program as having been aborted early by user request.

Unexpected PSS state Code nnn (-1110)

Explanation: The software has detected an unexpected change in the power sequencing control hardware. The code value indicates what state has been encountered. This error should never be seen, and may indicate a software or hardware problem on either the AWC or CIP.

User action: If safe to do so, attempt to enable HIGH POWER and note any different error messages which occur. If this error persists, contact Adept Customer Service.

Unexpected text at end of line (-451)

Explanation: The previous command or instruction cannot be recognized by eV+, possibly because of a mistyped function name or because an argument was specified where none is allowed.

User action: Reenter the line, correcting the syntax error.

Unexpected zero index Belt n (-1012)

Explanation: A zero index signal was received from the encoder for this motor belt at an unexpected time. The encoder may be gaining or losing counts, there may be a hardware problem with the zero index signal, or the Counts per zero index configuration parameter may be set incorrectly.

User action: Continue to use the system. Contact Adept Customer Service if this error occurs repeatedly.

Unexpected zero index Mtr n (-1005)

Explanation: A zero index signal was received from the encoder for this motor at an unexpected time. The encoder may be gaining or losing counts, there may be a hardware problem with the zero index signal, or the Counts per zero index configuration parameter may be set incorrectly.

User action: Turn on high power, calibrate the robot, and continue to use the system. If this error occurs repeatedly, contact Adept Customer Service.

Unknown editor command (-363)

Explanation: An unknown keystroke or extended command was issued while using the SEE program editor.

User action: Enter another command.

Unknown error code (-800)

Explanation: An error code that does not correspond to a known error message was received by eV+ from an external device.

User action: If an external computer is communicating with eV+ when the error occurs, verify that it is sending proper error codes. Otherwise, a software error is indicated. It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

Unknown function (-462)

Explanation: While accepting a program statement, eV+ has encountered a reference to a function that it does not recognize. This can be due to a mistyped function name or the leaving out of an operator between a symbol and a left parenthesis.

User action: Verify the spelling and syntax and reenter the line.

Unknown instruction (-452)

Explanation: An instruction was entered (or read from a disk file) that was not recognized by the system. This error is often caused by mistyping the instruction name, or trying to use a command as an instruction or vice versa. Note that statements with errors are turned into bad lines beginning with a question mark.

If the message occurred while loading a file from the disk, the file was probably created off-line, or with a different eV+ system (different version or options), and the indicated line is not compatible with the eV+ system in use.

User action: Correct the line or enter it again, making sure the spelling and usage are correct. When using the SEE editor, an invalid statement is either converted to a bad line or must be corrected before you can leave that line (depending on the setting of the AUTO.BAD feature). In the case of an error while loading from the disk, edit the program to correct the indicated instruction.

Unknown keyword (-424)

Explanation: The keyword in an FSET instruction is unknown in the context in which it was found. (Most often, a keyword used for a serial line was used when referencing a window or vice versa.)

User action: Correct the line in the executing program or reenter the command with the correct keyword.

Unknown network node (-537)

Explanation: A reference has been made to a network node address that is not known by the local network.

User action: Make sure that the correct node address was specified. make sure that the remote node is active and connected to the network. If explicit routing tables are used, make sure that they specify this node.

Variable type mismatch (-465)

Explanation: One or more of the variables in the line is of a type inconsistent with the other variables or with the type required by the command or instruction. For example, you may be trying to mix location variables with real-valued variables. If this error occurs upon exiting

from the editor, the variable type within the program conflicts with the type of a global variable that is already defined.

User action: Verify the syntax for the operation and reenter the line, correcting the mismatch. Delete conflicting global variables, if appropriate.

Warning Monitoring watchpoint (55)

Explanation: Program execution has begun while a watchpoint is set.

User action: None. This is an informational message. You may want to disable the watchpoint to eliminate its slowing down of program execution.

Warning Network monitor interface not open (103)

Explanation:eV+ cannot open a network port for use by ActiveeV+. This error should never be seen.

User action: Verify that the eV+ controller is properly connected to the network. Reboot eV+ and try again.

Warning Not calibrated (51)

Explanation: The robot servo system and joint position sensors are not calibrated. Thus, any location variables that are defined may not represent the locations desired.

User action: Enter a CALIBRATE command or have your program execute a CALIBRATE instruction.

Warning Protected and read-only programs are not stored (52)

Explanation: A STORE command has been executed while protected and/or read-only programs are loaded in the eV+ system memory. The protected and read-only programs are not stored in the new disk file.

User action: Use the FCOPY command if you want to move read-only programs from one disk to another. Protected programs cannot be moved from one disk to another.

Warning SET.SPEED switch disabled (54)

Explanation: A PRIME operation has been performed from the manual control pendant while the SET.SPEED system switch is disabled. Therefore, the monitor speed specified in the PRIME command has no effect.

User action: If you want the PRIME command to change the monitor speed, type the command **enable set.speed** at the keyboard.

Warning Watchdog timer disabled (56)

Explanation: Displayed at startup by all CPUs if the watchdog timer on the board is disabled. This timer is a hardware device that asserts SYSFAIL on the VME bus (which drops high power) if the CPU halts or gets hung. The board that has failed should light its SF LED. This message is also displayed whenever a user task is started from the monitor and the watchdog timer is disabled.

User action: Do not use this system. The watchdog timer must be enabled for safe operation of your system. The watchdog timer setting on the AWC cannot be changed by you. Report this problem to Adept Customer Service.

Watchpoint changed at (task) program_name, step n. ... ()

Explanation: A watchpoint has been enabled, and the watchpoint expression has changed.

User action: Continue debugging session.

Wrong disk loaded (-521)

Explanation: The diskette in a disk drive has been changed while a file was still open. Further attempts to access the file result in this error. Data being written into the file may be lost.

User action: Check your diskette to see if any data was lost. If so, it's too late now. Be more careful in the future.

---

[1] The command keys Ctrl+P, Ctrl+X, and Ctrl+Z are accepted only while using the eV+ program debugger in its monitor mode.

[2] The command keys Ctrl+P, Ctrl+X, and Ctrl+Z are accepted only while using the eV+ program debugger in its monitor mode.

SystemMessages - Numerical List

This section lists all the eV+ messages that have a numeric code. Most message codes associated with errors can be made available to a program by the ERROR function, which returns the code of the latest error that occurred. In addition, the $ERROR function returns the error message associated with any eV+ error code.

The information for each message below consists of the message code, the text of the message, and sometimes a comment about the applicability of the message. Angle brackets (<...>) are used to enclose a description of an item that appears in that position. All numbers are decimal.

The system messages are arranged numerically by system message code. For complete details on a message, click the message number to view the complete documentation. For a

list of the system messages sorted alphabetically by the first character of the message, see System Messages - Alphabetical List.

Message Numbering Convention

Informational Messages (numbers 0-49) lists messages that provide information.

Warning Messages (number 50-299) lists warning messages that you may receive.

Error Messages (negative numbers) lists the error messages that you may receive.

Informational Messages

Code	Message Text	Comments
0	Not complete	
1	Success	(General success response)
2	<no message>	(Signals start of program execution)
3	Program completed	
4	Program task # stopped at ...	
5	<no message>	(Signals start of DO processing)
6	<no message>	(Signals completion of DO processing)
7	<program instruction step>	(For TRACE mode of execution)
8	(HALTED)	
9	(PAUSED)	
10	Are you sure (Y/N)?	

Code	Message Text	Comments
11	Change?	
12	Okay to restart the servos (Y/N)?	
15	Program HOLD	
16	*Input error* Try again:	
17	Breakpoint at (task) program_name, step n	
18	Watchpoint changed at (task) program_name, step n Old value: n, New value: n	

Warning Messages

Code	Message Text
50	Executing in DRY.RUN mode
51	*Warning* Not calibrated
52	*Warning* Protected and read-only programs are not stored
53	*Protected program*
54	*Warning* SET.SPEED switch disabled
55	*Warning* Monitoring watchpoint
56	*Warning* Watchdog timer disabled
57	Press HIGH POWER button to enable power
58	Push, release, hold, MCP enable switch
60	Press HIGH POWER button when blinking

Code	Message Text
100	Network connection opened
101	Network connection closed
102	Network packet partially read
103	*Warning* Network monitor interface not open

Error Messages

Code	Message Text
-1	*Not yet implemented*
-2	*Obsolete keyword*
-300	*Illegal monitor command*
-301	*No program specified*
-302	*DO not primed*
-303	*Controller not in automatic mode*
-304	*Controller not in manual mode*
-305	*Storage area format error*
-307	*Program not executable*
-308	*Program interlocked*
-309	*Program already exists*
-310	*Can't access protected or read-only program*
-311	*Invalid when program task active*

Code	Message Text
-312	*Can't start while program running*
-313	*Can't go on, use EXECUTE or PRIME*
-314	*Switch can't be enabled*
-315	*Invalid software configuration*
-316	*Software checksum error*
-317	*Controller not in network mode*
-318	*Program task not active*
-319	*Program task not in use*
-350	*Can't delete .PROGRAM statement*
-351	*First statement must be .PROGRAM*
-352	*Invalid in read-only mode*
-353	*No other program referenced*
-354	*Line too long*
-355	*Can't exit while lines attached*
-356	*Not found*
-357	*Recursive macros illegal*
-358	*Canceled*
-359	*Illegal in debug monitor mode*
-360	*Must be in debug mode*
-361	*Can't change modes while task running*

Code	Message Text
-362	*Can't execute from SEE program instruction*
-363	*Unknown editor command*
-364	*Can't create program in read-only mode*
-365	*Illegal in read-write mode*
-366	*Invalid when program on stack*
-380	*Breakpoint not allowed here*
-400	Aborted
-401	*Undefined value*
-402	*Illegal value*
-403	*Illegal assignment*
-404	*Illegal array index*
-405	*Illegal digital signal*
-406	*Undefined program or variable name*
-407	*Invalid argument*
-408	*Program argument mismatch*
-409	*Arithmetic overflow*
-410	*Negative square root*
-411	*Not enough storage area*
-412	*Branch to undefined label* Step nnn
-413	*Not enough program stack space*

Code	Message Text
-414	*Can't mix MC & program instructions*
-416	*String variable overflow*
-417	*String too short*
-418	*Illegal memory reference*
-419	*Illegal when command program active*
-420	*Undefined value in this context*
-421	*Program not on top of stack*
-422	*Function already enabled*
-423	*Illegal operation*
-424	*Unknown keyword*
-425	*Calibration program not loaded*
-426	*Can't find calibration program file*
-450	*Can't interpret line*
-451	*Unexpected text at end of line*
-452	*Unknown instruction*
-453	*Ambiguous name*
-454	*Missing argument*
-455	*Invalid program or variable name*
-456	*Invalid number format*
-457	*Reserved word illegal*

Code	Message Text
-458	*Illegal expression syntax*
-459	*Missing parenthesis*
-460	*Missing quote mark*
-461	*Invalid format specifier*
-462	*Unknown function*
-463	*Invalid statement label*
-464	*Duplicate statement label*
-465	*Variable type mismatch*
-466	*Illegal use of belt variable*
-467	*Illegal .PROGRAM statement*
-468	*Duplicate .PROGRAM arguments*
-469	*Attempt to redefine variable type*: variable_name
-470	*Attempt to redefine variable class*: variable_name
-471	*Misplaced declaration statement*
-472	*Control structure error* Step nnn
-473	*Control structure error*
-474	*Too many array indices*
-475	*Missing bracket*
-476	*Invalid qualifier*

Code	Message Text
-477	*Ambiguous AUTO invalid*
-500	*File already exists*
-501	*Nonexistent file*
-502	*Illegal I/O device command*
-503	*Device full*
-504	*Unexpected end of file*
-506	*File already open*
-507	*I/O communication error*
-508	*Device not ready*
-509	*Directory error*
-510	*Data checksum error*
-511	*Input block error*
-512	*File format error*
-513	*File not opened*
-514	*File or subdirectory name error*
-515	*Already attached to logical unit*
-516	*Not attached to logical unit*
-517	*I/O queue full*
-518	*Illegal I/O channel number*
-519	*Driver internal consistency error*

Code	Message Text
-520	*Invalid disk format*
-521	*Wrong disk loaded*
-522	*Data error on device*
-523	*Bad block in disk header*
-524	*Communications overrun*
-525	*Illegal I/O redirection specified*
-526	*No data received*
-527	*Illegal user LUN specified*
-528	*Illegal record length*
-529	*Output record too long*
-530	*Protection error*
-531	*Communication time-out*
-532	*Out of I/O buffer space*
-533	*Invalid hardware configuration*
-534	*Network restarted remotely*
-535	*Network closed locally*
-536	*Too many network errors*
-537	*Unknown network node*
-538	*Network node off line*
-539	*No matching connection*

Code	Message Text
-540	*Invalid connection specified*
-541	*Invalid network protocol*
-542	*Network not enabled*
-543	*Illegal when network enabled*
-544	*Not configured as accessed*
-545	*Nonexistent subdirectory*
-546	*Subdirectory list too long*
-547	*Subdirectory in use*
-548	*Illegal while protocol active*
-549	*Out of graphics memory*
-550	*Too many windows*
-551	*Font not loaded*
-552	*Graphics processor timeout*
-553	*Too many arguments*
-554	*Too many closeable windows*
-555	*Graphics storage area format error*
-557	*Can't create new slide bar*
-558	*Graphics software checksum error*
-559	*Out of network resources*
-560	*Invalid network resource*

Code	Message Text
-561	*Invalid network address*
-562	*Network timeout*
-563	*Remote has not exported network resource*
-564	*Network resource name conflict*
-565	*Network connection terminated*
-566	*Not owner*
-567	*Not a directory*
-568	*Is a directory*
-569	*File too large*
-570	*File name too long*
-571	*Directory not empty*
-587	*Network resource already in use*
-600	*Stopped due to servoing error*
-601	*Robot not attached to this program*
-602	*Robot already attached to program*
-603	*COMP mode disabled*
-604	*Robot power off*
-605	*Robot not calibrated*
-606	*Robot overheated*
-607	*No air pressure*

Code	Message Text
-608	*External E-STOP 1 detected by CPU*
-609	*Illegal joint number*
-610	*Location out of range*
-611	*Must use straight-line motion*
-612	*Straight-line motion can't alter configuration*
-613	*Illegal motion from here*
-614	*Attempt to modify active belt*
-615	*Belt not enabled*
-616	*Belt window violation*
-617	*Belt servo dead*
-618	*Location too close*
-619	*Invalid orientation*
-620	*Speed pot or STEP not pressed*
-621	*Robot interlocked*
-622	*No robot connected to system*
-623	*Stop-on-force triggered*
-624	*Force protect limit exceeded*
-625	*Invalid servo initialization data*
-626	*Can't ALTER and track belt*
-627	*Robot power on*

Code	Message Text
-628	*Robot module not loaded* ID:n
-629	*SYSFAIL detected by CPU*
-630	*Backplane E-STOP detected by CPU*
-632	*Power failure detected by robot*
-633	*PANIC command*
-635	*Cartesian control of robot not possible*
-636	*Trajectory clock overrun*
-637	*Illegal while joints SPIN'ing*
-638	*SPIN motion not permitted*
-639	*Manual brake release*
-640	*E-STOP from robot*
-641	*E-STOP from amplifier*
-642	*SYSFAIL detected by robot*
-643	*E-STOP detected by robot*
-644	*Incompatible safety configuration*
-645	*Power disabled: Manual/Auto changed*
-646	*HIGH POWER button not pressed*
-647	*Collision avoidance dead-lock*
-648	*Safety system not commissioned*
-649	*Timeout: MCP enable switch not toggled*

Code	Message Text
-650	*Manual control pendant failure*
-651	*RSC communications failure*
-652	*RSC reset*
-653	*RSC time-out*
-654	*RSC transmission garbled*
-655	*RSC bad packet format*
-656	*RSC calibration load failure*
-657	*Pendant Not Connected*
-658	*Device hardware not present*
-659	*Device time-out*
-660	*Device error*
-661	*NVRAM data invalid*
-662	*Device sensor error*
-663	*Device reset*
-665	*NVRAM battery failure*
-666	*Must use CPU #1*
-667	*Power failure detected*
-668	*Device in use*
-669	*RSC hardware failure*
-670	*RSC power failure*

Code	Message Text
-671	*Servo board 12V fuse open*
-672	*Servo board solenoid fuse open*
-673	*Servo board E-Stop fuse open*
-674	*Servo task overloaded*
-675	*Timeout enabling power*
-676	*RSC module ID doesn't match robot*
-677	*Servo protocol incompatible*
-678	*Duplicate servo node ID*
-679	*Expected servo node not found*
-680	*Servo node not downloaded*
-800	*Unknown error code*
-804	*Option not installed*
-805	*Hardware not in system*
-859	*Database manager internal error*
-900	*Robot module not enabled*
-901	*Obstacle collision detected*
-902	*Interrupted multi-segment motion*
-903	*DeviceNet: Critical device off-line*
-904	*[Fatal] E-STOP signals are stuck off*
-905	*[Fatal] I/O processor failure*

Code	Message Text
-906	*Robot power off requested*
-907	*E-STOP circuit relay failure*
-908	*E-STOP from front panel button*
-909	*E-STOP from MCP E-STOP button*
-910	*E-STOP from user E-STOP button*
-911	*E-STOP from front panel external input*
-912	*E-STOP from user enable switch*
-913	*E-STOP from MCP enable switch*
-914	*E-STOP 2 detected by CPU*
-915	*MCP enable switch 1 off detected by CPU*
-916	*MCP enable switch 2 off detected by CPU*
-917	*Manual mode switch 1 off detected by CPU*
-918	*Manual mode switch 2 off detected by CPU*
-919	*E-STOP asserted by CPU*
-920	*[Fatal] Manual mode switch stuck off*
-921	*E-STOP from user muted safety gate*
-922	*E-STOP channels 1 and 2 do not match*
-923	*E-STOP circuit is shorted*
-924	*Front panel HIGH POWER lamp failure*

Code	Message Text
-925	*Front panel serial bus failure*
-927	*1394 communications timeout*
-929	*E-Stop from Line E-Stop input*
-931	*Servo hardware failure*
-932	*Manual mode cannot be enabled*
-935	*Orientation out of range*
-936	*Kinematic solution not found*
-937	*Robot already under manual control*
-938	*Joint control of robot not possible*
-999	Aborted
-1001	*Invalid servo error* Mtr n
-1002	*Position out of range* Jt
-1003	*Time-out nulling errors* Mtr n
-1004	*No zero index* Mtr n
-1005	*Unexpected zero index* Mtr n
-1006	*Soft envelope error* Mtr n
-1007	*Motor stalled* Mtr n
-1008	*Encoder quadrature error* Mtr n
-1009	*Timeout enabling amplifier* Mtr n
-1010	*Invalid error code* Belt n

Code	Message Text
-1011	*No zero index* Belt n
-1012	*Unexpected zero index* Belt n
-1013	*Encoder quadrature error* Belt n
-1014	*[Fatal] Initialization failure* Mtr n
-1015	*Initialization failure* Belt n
-1016	*Motor overheating* Mtr n
-1018	*Motor amplifier fault* Mtr n
-1021	*Duty-cycle exceeded* Mtr n
-1022	*Skew envelope error* Mtr n
-1023	*Position out of range* Mtr n
-1025	*Encoder fault* Mtr n
-1026	*Software incompatible* Code n
-1027	*Hard envelope error* Mtr n
-1028	*Soft overspeed error* Mtr n
-1029	*Negative overtravel* Mtr n
-1033	*Positive overtravel* Mtr n
-1034	*Overtravel* Mtr n
-1035	*Safety speed limit error* Chn nn
-1101	*[Fatal] Servo process dead* CPU n
-1102	*[Fatal] Servo code incompatible* CPU n

Code	Message Text
-1104	*[Fatal] Servo dead* Mtr n
-1105	*Motor startup failure* Mtr n
-1106	*Calibration sensor failure* Mtr n
-1107	*[Fatal] Servo init failure* CPU n
-1110	*Unexpected PSS state* Code nnn
-1111	*E-STOP from safety system* Code n
-1112	*Remote digital input failed* Block N
-1113	*Remote digital output failed* Block N
-1115	*Robot power system failure* Code n